5G RAN/MEC Slicing and Admission Control using Deep Reinforcement Learning

by

Arash Moayyedi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

Statement of Contributions

Chapters 1 and 2 borrow content and figures from the papers: “Multi-Agent Deep Reinforcement Learning for Slicing and Admission Control in 5G C-RAN”[1], and “Coordinated Slicing and Admission Control using Multi-Agent Deep Reinforcement Learning” [2].

Abstract

The 5G RAN functions can be virtualized and distributed across the radio unit (RU), distributed unit (DU), and centralized unit (CU) to facilitate flexible resource management. Complemented by multi-access edge computing (MEC), these components create network slices tailored for applications with diverse quality of service (QoS) requirements. However, as the requests for various slices arrive dynamically over time and the network resources are limited, it is non-trivial for an infrastructure provider (InP) to optimize its long-term revenue from real-time admission and embedding of slice requests. Prior works have leveraged Deep Reinforcement Learning (DRL) to address this problem, however, these solutions either do not scale to realistic topologies, require re-training of the DRL agents when facing topology changes, or do not consider the slice admission and embedding problems jointly. In this thesis, we use multi-agent DRL and Graph Attention Networks (GATs) to address these limitations. Specifically, we propose novel topology-independent admission and slicing agents that are scalable and generalizable to large and different metropolitan networks. Results show that the proposed approach converges faster and achieves up to 35.2% and 20% gain in revenue compared to heuristics and other DRL-based approaches, respectively. Additionally, we demonstrate that our approach is generalizable to scenarios and substrate networks previously unseen during training, as it maintains superior performance without re-training or re-tuning. Finally, we extract the attention maps of the GAT, and analyze them to detect potential bottlenecks and efficiently improve network performance and InP revenue through eliminating them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The decomposition of the Fifth Generation (5G) radio access network (RAN) plays a key role in the 5G New Radio (NR) architecture. The RAN protocol stack, *i.e.*, baseband unit (BBU) functions in 4G, can be split into radio unit (RU), distributed unit (DU), and central unit (CU), which provides flexibility in distributing network functions across the RAN to meet latency and bandwidth requirements of varying services. Multi-access edge computing (MEC) can further facilitate stringent quality of service (QoS) guarantees by placing application servers close to end-users. By leveraging Network Function Virtualization (NFV), the 5G infrastructure provider (InP) can cater to a variety of MEC-enabled use-cases (*e.g.*, cloud gaming, remote surgery) with variable QoS requirements, providing end-to-end (E2E) isolated and differentiated networks for each application, *i.e.*, *network slices*. However, to achieve efficiency and revenue gains from network slicing, coordinated and dynamic management of the entire 5G system is imperative, including the RAN, MEC, core, and the transport networks connecting them.

Metropolitan 5G networks usually consist of multi-level nodes—access, aggregation, and core—interconnected by mesh-like or multi-ring topologies [8, 9]. Connected to the cell sites are the access nodes, which are linked to the core by aggregation nodes. Centralized sites and links offer more resources, but incur more transport latency and bandwidth as unprocessed data travels to the higher-layer sites. From the InP's perspective, centralizing RAN and MEC virtual network functions (VNFs) can increase multiplexing gains (*e.g.*, power consumption and maintenance [10]), and prevent computing bottlenecks at the access, provided that the delay constraints and bandwidth requirements of individual VNFs are met. Therefore, to ensure the optimal placement of VNFs, technical and cost-effective trade-offs between throughput, latency, and centralization must be considered.

MEC is best suited to services requiring very high bandwidth and ultra-low latency, such as virtual reality and mission-critical applications. Requests for these services are generated dynamically over time by *slice tenants*, such as service providers. Slices consist of RAN (*i.e.*, RU, DU, CU) and MEC VNFs, along with virtual links (VLs) that connect them, collectively referred to as Virtual Networks (VNs). However, the addition of MEC complicates the RAN slicing problem by imposing the E2E MEC service latency requirements on top of the latency requirements of individual RAN components. InPs generate revenue from accepting and accommodating network slice requests, but not all requests can be met due to resource limitations in the substrate network. Moreover, the revenue earned for each request may vary depending on factors such as the tenant's subscription and QoS requirements. Therefore, to maximize the long-term revenue, the InP should: i) perform admission control (AC) in a way that maximizes the long-term revenue, ii) place the RAN and MEC VNFs according to their delay and processing requirements, and iii) route traffic from the originating cell site to the core, which is the Internet's interface to the mobile infrastructure. We refer to the latter two (*i.e.*, ii and iii) decisions as slicing.

1.1 Motivation

The problem of RAN/MEC slicing has previously been investigated in an offline manner [11, 12, 13], *i.e.*, assuming all slice requests are known in advance. Recently, deep reinforcement learning (DRL) has shown promising performance in the RAN slicing [14] and AC problems [15, 16] in an online, but disjoint setting. Sulaiman et al. [1] designed a joint AC and RAN slicing solution and showed that a joint solution not only makes for a more practical 5G slice orchestration scenario, but that it also benefits the InPs in terms of the long-term revenue. They designed their solution based on a simple multi-tier network using multi-agent DRL (MADRL) and demonstrated its superior performance over its single-agent counterpart. However, they use multi-layer perceptron (MLP) models for both slicing and AC, which limits scalability and generalizability across different networks.

With MLP architectures, information about the network state, which includes the topology, and features of all nodes and links, is passed to the model as a whole, *i.e.*, as a flattened specifically ordered vector. Moreover, the formats of these feature vectors are defined during model initialization and training. As a result, with the slightest topological variation, *e.g.*, node/link failure or network expansion, the model becomes obsolete and requires re-training or re-tuning. The state-of-the-art DRL-based approaches for slicing often rely on MLP-based architectures [14, 17, 18] which fail to operate on previously unseen topologies and require training the model from scratch. Training a deep neural network from scratch can be prohibitive, especially in a highly dynamic environment, or under tight service delay requirements. In addition, it is also infeasible to train separate models for all possible topology variations, as it would lead to an exponential number of models.

1.2 Contributions

In this work, we integrate multi-agent DRL with Graph Neural Network (GNN), which provides promising topology-independent feature extraction capabilities [19]. More specifically, we use a recent variant of the popular Graph Attention Networks (GATs) [20], GATv2 [21], which is a spatial-based GNN model (*cf.*, Section 2.3.1). Spatial-based methods are popular for their efficiency, flexibility, and generalizability and work by propagating node features across edges [19]. In GATs, an attention mechanism is used that effectively exploits the structural characteristics of networks by learning how each node is influenced by its neighbours. In addition, different from all other GNNs, they can take the edge features directly into account. Utilizing GATv2 and a novel learning model, we propose a generalizable RAN/MEC slicing DRL agent. For AC, we employ an MLP with a fixed-size input that consists of features of the 4 nodes and edges where VNFs (*i.e.*, RU, DU, CU, and MEC) and VLs are respectively placed, and the slice request information. Such input is independent of the topology, so the MLP-based AC agent is able to operate in presence of changing network conditions.

The main contributions of this thesis are:

- The problem of joint AC and MEC/RAN slicing in 5G metropolitan networks under E2E service delay and resource constraints is modeled as an integer linear programming (ILP) problem and proven to be NP-hard.
- A novel solution for the joint online AC and slicing problem is proposed using multi-agent DRL. For slicing, a generalizable GNN-based DRL agent, and for AC, a topology-independent MLP is devised, which allows both of them to operate on arbitrary topologies.
- The proposed approach is evaluated and compared to greedy and state-of-the-art heuristics, and DRL-based solutions. Our model outperforms the baselines by up to 35.2% in the overall revenue gain.

- The robustness and generalization of our slicing and AC agents are evaluated under varying network conditions. The agents outperform other heuristic approaches by up to 25.5% even in large-scale previously unseen network topologies.
- The attention maps of different topologies are extracted from the GNN-based deep model of the slicing agent. It is shown that these maps can be utilized to identify bottlenecks and improve capacities of select nodes to efficiently increase InP revenue.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide a brief background on the problems of RAN slicing and admission control and go through the technical details of deep reinforcement learning and graph neural networks. We then review the state of the art that inspired this work and discuss the challenges that were present when designing the solution. Chapter 3 explores the system model of RAN slicing and mathematically formulates the joint AC and MEC/RAN slicing problem. The proposed method is introduced in Chapter 4 and in Chapter 5, we evaluate it and provide insight into its advantages over the existing methods. Finally, Chapter 6 concludes the thesis and introduces possible research extension directions.

Chapter 2

Background

2.1 Slicing and Admission Control

The 5G RAN is responsible for providing the connection between the mobile device and the core network and comprises chains of network functions (*e.g.*, High-PHY, Low-PHY, etc.) that belong to the NR protocol stack [3]. These functions perform a variety of tasks, such as managing the allocation of radio resources in the network, including the allocation of spectrum and the assignment of frequencies to different devices, and managing the movement of mobile devices within the network, including handovers between base stations and the tracking of devices. In legacy RAN architectures, these functions were placed statically, which could potentially lead to bottlenecks and under-utilization of the network infrastructure. However, with a greater emphasis on virtualization and the use of software-defined networking (SDN) technologies, and the adoption of Cloud RAN (C-RAN) in 5G mobile networks, the substrate network has been re-imagined as a network of interconnected sites, each consisting of a number of commodity servers or nodes. NFV allows an InP to virtualize these resources and facilitates flexible and strategic placement of the VNFs at different sites. This move from a conventional one-size-fits-all network infrastructure towards more flexible networks allows for intelligent decision making through analyzing the network trends. This results in a minimization of bottlenecks, maximization of infrastructure utilization, and cost savings by reducing hardware procurement and maintenance, leading to a higher InP revenue.

In a metro 5G RAN, the interconnected sites can form a variety of shapes, based on the requirements. In a ring topology, nodes (*e.g.*, base station or access point) are arranged in a circular configuration, with each node connecting to two other stations. This creates

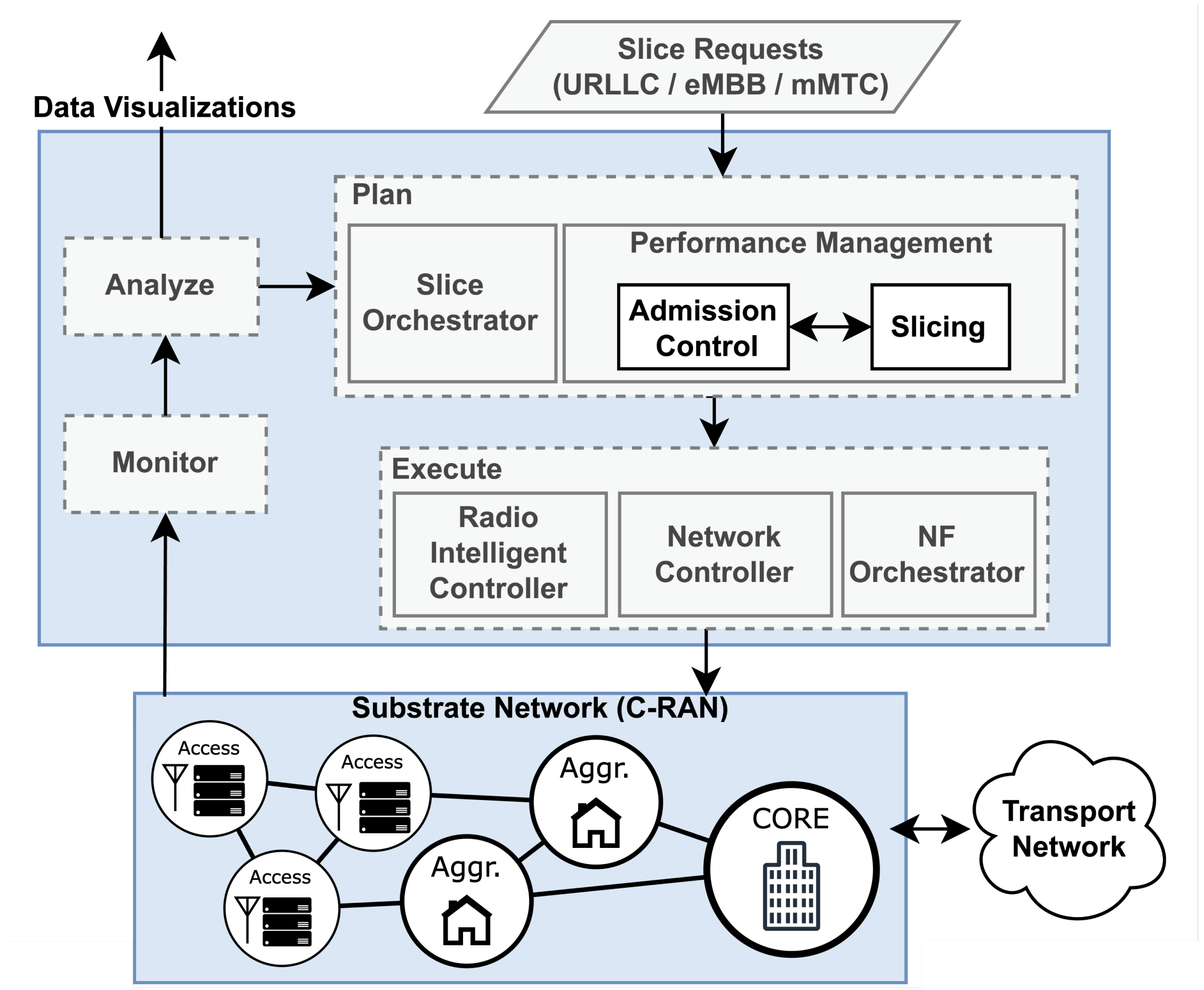


Figure 2.1: High-level view of closed-loop, autonomous management and orchestration of VNFs in 5G C-RAN

a closed loop of stations that allows data to be transmitted in a continuous loop. A spur topology is similar to a ring topology, but with one or more sites that are connected to the ring at a single point. A multi-ring RAN topology consists of multiple rings of nodes connected together, which allows for greater coverage and capacity, as data can be transmitted among nodes on different rings as well as within a single ring. One advantage of ring and spur topologies is that they are relatively simple and easy to set up, as there are fewer connections to be made between the nodes. However, they can be less resilient to failures, as a break in the ring or spur can disrupt communication for all the nodes in the network. In contrast, mesh topologies, where each node is connected to multiple other nodes, are generally more resilient to failures, as data can be transmitted between each pair of nodes using multiple paths.

The 5G mobile networks are poised to support a wide range of services, primarily categorized into enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communications (URLLC), and massive Machine-Type Communications (mMTC), based on their QoS requirements (*e.g.*, bandwidth, latency and mobility). eMBB slices are designed to support high-bandwidth applications, such as streaming video and online gaming. They typically require high capacity and are optimized for high data rates. URLLC slices are designed to support real-time applications that require ultra-low latency and high reliability, such as remote surgery and autonomous driving. mMTC slices are designed to support the communication needs of a large number of machine-type devices, such as sensors and smart meters. They are optimized for low data rates and long battery life. Network slicing is a key enabling technology to offer isolated E2E virtual networks in 5G, that are tailored to satisfy the specific QoS requirements of different services on the same infrastructure. Network slices can include chains of RAN and core VNFs. The placement (*i.e.*, Virtual Network Embedding (VNE)) of RAN VNFs in 5G C-RAN should consider the service type and its Service-Level Agreements (SLAs).

Having MEC allows computing resources to be placed at the edge of the network, closer to users and devices. This can improve the performance and latency of applications and services that rely on large amounts of data processing, such as augmented reality, virtual reality, and real-time analytics. In the context of RAN slicing, MEC can be used to create slices that are optimized for specific services that require low latency and high-bandwidth connectivity, for a smooth and immersive user experience. MEC can be implemented in a number of different ways, including using edge servers, edge clouds, or edge data centers. By bringing computing resources closer to the network edge, MEC can help to reduce the amount of data that needs to be transmitted over long distances, improving performance and quality of experience. Moreover, it can help to offload processing from the core network, improving overall network efficiency.

Accepting a slice-request (SR) contributes to the InP's revenue. However, given an InP's limited resources, it is impossible to serve all incoming SRs. Additionally, the amount of revenue that SRs bring may also vary (*e.g.*, based on their priority or QoS requirements). Therefore, an AC decision must be made for each incoming SR, such that it maximizes the InP's long-term revenue. Since, in practice, the slice-request traffic is not known in advance, there is also a need for algorithms that can predict future slice-request traffic based on the past traffic and make intelligent slice admission decisions based on these predictions. Reinforcement Learning (RL) can learn to optimize a given objective even without a comprehensive system model. This makes RL combined with Deep Neural Networks (DNNs) particularly suitable for these types of problems with complex system models.

For the system design, in this work, we consider the MAPE (*i.e.*, monitor, analyze, plan, execute) control loop [22, 23] to facilitate closed-loop, autonomous management and orchestration of VNFs in 5G C-RAN, as depicted in Fig. 2.1. The monitor module intelligently collects data from the substrate network and sends it to the analyze module. From raw data, the analyze module extracts useful information and computes various metrics required for visualization and planning (*e.g.*, QoS, network infrastructure state). The processed data and information regarding incoming SRs are received by the plan module.

The plan module performs intelligent slice orchestration and performance management using AI/ML techniques [24]. The proposed intelligent AC and slicing schemes in this thesis are sub-components of the performance management component, which are responsible for the admission and embedding of network slices. These sub-components can be employed either concurrently (*i.e.*, both output their decisions independently) or sequentially (*i.e.*, each sub-component can use the output of the other to make its decision). If an SR is admitted, the slice orchestrator passes the appropriate commands (*e.g.*, instantiation of VNFs, links) to the execute module which in turn directs VNF orchestrator, network controller and Radio Intelligent Controller (RIC) components to set up the virtual machines, transport paths, and RAN radio resources in the substrate network, respectively. The RIC, introduced and standardized by the O-RAN Alliance [25], provides advanced control and configuration functionality for efficient management of RAN infrastructure. In this work, we simulate a substrate network, so the monitor, analyze and execute modules do not present a research challenge, and we only describe the pertinent system design of the AC and slicing components.

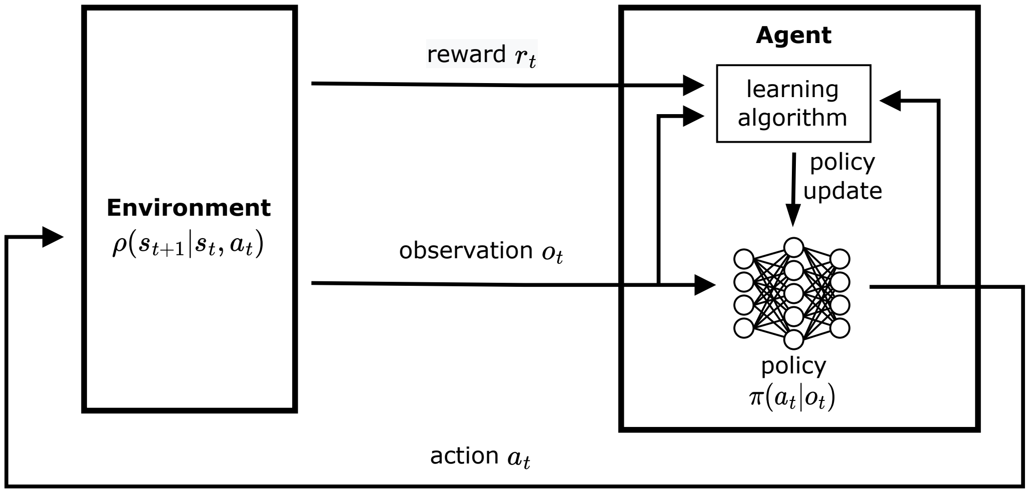


Figure 2.2: DRL architecture

2.2 Deep Reinforcement Learning

In reinforcement learning, an agent interacts with an environment to learn a policy that maximizes the expected cumulative reward by proposing actions [26]. The interaction between the RL agent and the environment can formally be described using a markov decision process (MDP). The agent-environment interaction in this thesis spans an infinite horizon, *i.e.*, the agent acts continuously. Therefore, we consider infinite horizon MDP defined by the tuple $(O,\ A,\ r,\ \rho,\ \rho_0,\ \gamma)$, where O is the state space, A is the action space, $r : O \times A \times O \to \mathbb{R}$ is the reward function, $\rho : O \times A \times O \to [0, 1]$ is the state transition probability distribution, ρ_0 is the initial state distribution, and γ is the discount factor. The aim of the RL agent is to learn either a deterministic policy $\pi : O \to A$ or a stochastic policy $\pi : O \times A \to [0, 1]$ that maximizes the expected discounted return given as $G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r_k$, where r_k is the reward at time step k.

For policy π, the state value function V_π, state-action value function Q_π, and the advantage function A_π are defined as [26]:

$$V_\pi(o) = \mathbb{E}_\pi \left[G_t \mid o_t = o \right], \forall o \in O, \tag{2.1}$$

$$Q_\pi(o, a) = \mathbb{E}_\pi \left[G_t \mid o_t = o, a_t = a \right], \forall o \in O,\ \forall a \in A, \tag{2.2}$$

$$A_\pi(o, a) = Q_\pi(o, a) - V_\pi(o),\ \forall o \in O,\ \forall a \in A, \tag{2.3}$$

i.e., the expected return of starting from state o in the case of $V_\pi(o)$, and starting from state o and taking the action a in the case of $Q_\pi(o,a)$. $A_\pi(o,a)$ measures the relative state-action value of taking action a in state o as compared to the state value of state o. The state-value and state-action value functions are collectively referred to as value functions.

When the number of possible states and actions is small, a tabular method can be used to store the value functions and derive an effective policy. However, this method becomes inefficient as the size of the state and action space increases. DRL approximates these tables using deep neural networks, which have the ability to generalize to previously unseen states, while involving a relatively smaller number of learnable parameters. Fig. 2.2 provides an overview of DRL, where the agent interacts with the environment by sending action a_t inferred from running policy π at time t, and receiving reward r_t and observation o_t. a_t, r_t, and o_t are then used by the agent to make updates to the policy network π.

DRL policy optimization algorithms in the literature are mainly divided into 3 categories: (i) value-based, (ii) policy-based, and (iii) Actor-Critic. Value-based methods entail learning either of the value functions and leveraging it to derive an optimal policy. Policy-based methods, on the other hand, can directly learn the optimal policy. Finally, Actor-Critic methods enfold learning both a value function (*i.e.*, Critic) and policy (*i.e.*, Actor) and have been shown to lead to faster empirical convergence [26]. In the thesis we employ the Proximal Policy Optimization (PPO) Actor-Critic method as our DRL algorithm, which we briefly discuss in the next section.

2.2.1 Proximal Policy Optimization

Let π_θ denote the stochastic policy in DRL, *i.e.*, the Actor neural network parameterized by the weights θ. Similarly, v_ϕ denotes the value function, *i.e.*, the Critic neural network parameterized by the weights ϕ. Policy-based methods learn neural network parameters using optimization methods such as policy-gradient. Line-search methods and trust-region methods present two distinct classes of such optimization methods. With line-search methods, the policy is improved by taking a step in the direction of the gradient that leads to a maximum increase in the objective function. With trust-region methods, on the other hand, a surrogate objective function is computed using a linear or quadratic approximation of the actual objective function. But these approximations are only accurate within a 'trust-region'. Therefore, trust-region methods optimize the policy parameters by iteratively approximating the objective function and finding the optimal parameters for that surrogate objective while staying within the trust-region. Trust-region policy optimization (TRPO) [27] and proximal policy optimization (PPO) [**?**] are two of the well-known trust-region methods in this area. TRPO formulates the policy update as a constrained optimization problem:

$$\begin{aligned} \theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}^{\pi_{\theta_k}}(s_t, a_t) \right] \\ \text{s.t.} \quad & \hat{\mathbb{E}}_t \left[KL \left[\pi_{\theta_k}(.|s_t), \pi_\theta(.|s_t) \right] \right] \leq \delta \end{aligned} \tag{2.4}$$

where θ_k and θ are the policy parameters before and after the update, respectively, KL is the Kullback–Leibler divergence, and δ is the hyperparameter that defines the size of the trust-region in terms of the KL-divergence between the two policies. Here, $\hat{\mathbb{E}}$ denotes the empirical expectation and $\hat{A}_t$ is the empirical advantage function. $\hat{A}_t$ can be estimated using Generalized Advantage Estimator (GAE) [**?**]. **?**] modify the TRPO algorithm and propose the PPO algorithm. They show that the constrained optimization problem in 2.4 can be replaced by truncating the new to old policy ratio to avoid deviating too much from the old policy.

In PPO, at each training iteration k, first the set of trajectories $\mathcal{D}_k = \{m_i\}$, where $m_i = \{s_0, a_0, r_0, \cdots, s_T, a_T, r_T\}$ are collected by running action a_t sampled from policy π_{θ_k} at state s_t and receiving reward r_t and next state s_{t+1}. Then, the parameters of the actor (θ) and critic (ϕ) networks are updated as:

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}} \frac{1}{|\mathcal{D}_k|T} \sum_{m \in \mathcal{D}_k} \sum_{t=0}^{T} \min(r_t(\theta)\hat{A}^{\pi_{\theta_k}}(s_t, a_t), clip(\epsilon, r_t(\theta))\hat{A}^{\pi_{\theta_k}}(s_t, a_t)), \tag{2.5}$$

$$\phi_{k+1} = \underset{\phi}{\operatorname{argmin}} \frac{1}{|\mathcal{D}_k|T} \sum_{m \in \mathcal{D}_k} \sum_{t=0}^{T} \left(v_\phi(s_t) - \hat{R}_t \right)^2, \tag{2.6}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$ and $\hat{R}_t$ is the discounted cumulative reward. The authors show that this unconstrained optimization step leads to a faster empirical convergence and better overall performance.

2.2.2 Multi-Agent Deep Reinforcement Learning

In multi-agent DRL (MADRL), multiple DRL agents act in a shared environment to maximize the long-term return. Agents can be designed to operate cooperatively, competitively, or in a mixed setting, based on the training paradigm. In independent training, each agent is trained independently using a separate reinforcement learning algorithm. This can be useful when the agents are not interacting with each other or when the interactions are relatively simple. Cooperative training is used when the agents are trained to work together to achieve a common goal, through designing a shared reward function that encourages the agents to collaborate, or using techniques such as centralized training with decentralized execution. In the competitive training paradigm, the agents are trained to compete with each other in order to maximize their own rewards. In this method, each agent's reward is against the other agent's reward, resulting in a rivalry among the agents. For mixed training, the agents are trained to both cooperate and compete with each other in order to achieve a complex goal. For instance, in a game with multiple players, the agents may need to cooperate in order to defeat a common enemy, but also compete with each other for resources or points. This can involve using a centralized critic network to evaluate the value of different actions taken by the agents, and decentralized actor networks to predict the best actions to take. The choice of training paradigm depends on the specific goals and constraints of the problem, as well as the characteristics of the agents and the environment. In this work, we aim to train both agents to maximize the InP's revenue by admitting more and higher-priority slices. Therefore, we utilize the cooperative training paradigm and shape the reward function to converge to the same objective.

Multi-agent settings often violate the fundamental assumptions underlying the theoretical foundation of single-agent RL that are necessary to guarantee convergence [28]. For example, when multiple RL agents are concurrently learning and acting in a common environment, the environment becomes non-stationary from the perspective of any individual agent. This can prevent the agents' policies from converging towards the optimal even when the goals of different agents are aligned.

2.3 Graph Neural Networks

Graph data differ from other types of data in several aspects, as they present a non-Euclidean data structure. For instance, Convolutional Neural Networks (CNNs) consider the correlation among adjacent pixels in an image by passing a kernel across it. However, images are comprised of a fixed grid of pixels that do not change throughout the data. Such a method cannot be applied to graphs, due to their complexity and dynamicity. GNNs are neural models designed to operate directly on graph-structured data and have numerous variants. The main goal of a GNN is to learn a low-dimensional vector representation for each node h_v, which can be used for different learning tasks. ConvGNNs are a popular variant of GNNs that extend the convolutional operator to graphs, motivated by the success of the CNN in computer vision. ConvGNNs stack multiple graph convolutional layers to extract high-level representations and have two categories: spectral-based and spatial-based [19].

The first category, including the popular Graph Convolutional Networks (GCN) [29], perform graph convolutional operation on the entire graph at once in the Fourier domain and thus suffers from poor scalability and generalizability. Spatial-based GNNs address these limitations by implementing graph convolutions leveraging a message-passing technique among the neighboring nodes of the graph to learn their relationship. As a result, they are able to capture the spatial correlation present in different parts of the graph, irrespective of its structure. In recent years, numerous variations of such architecture have been proposed.

In spatial-based methods, input graph nodes' representations $\left\{h_v \in \mathbb{R}^F \,|\, v \in \mathcal{V}\right\}$ are aggregated with those of their neighbours. The combined representations are then passed through a transformation function g (*e.g.,* a dense layer and a non-linearity) to output new representations $\left\{h'_v \in \mathbb{R}^{F'} \,|\, v \in \mathcal{V}\right\}$. Specifically, for each node, we perform

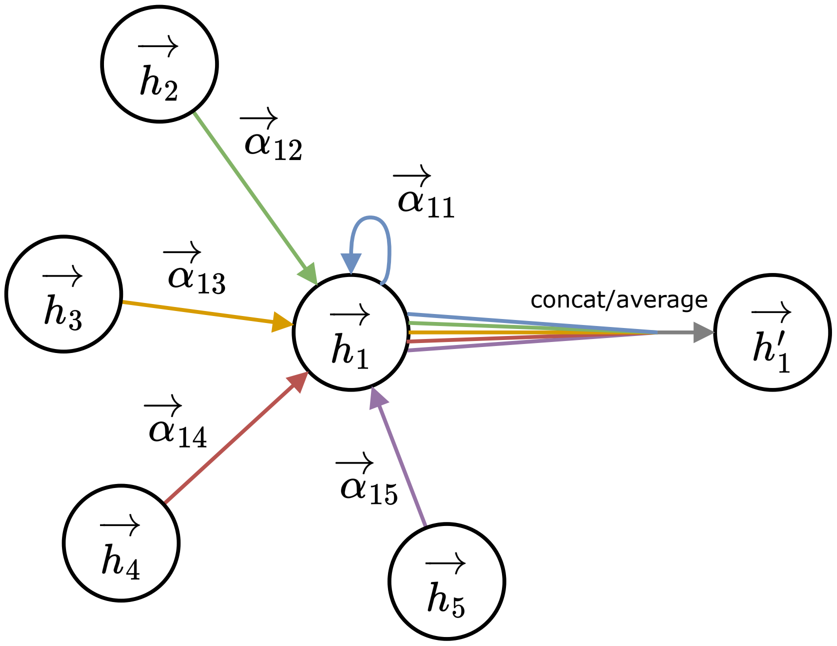
$$h'_v = g\left(h_v, \texttt{AGGREGATE}\left(\{h_u \,|\, u \in \mathcal{N}_v\}\right)\right), \tag{2.7}$$

where $\mathcal{N}_v$ is the set of neighbours of node v and `AGGREGATE` can be any permutation invariant function, *e.g.*, mean. The selection of g and `AGGREGATE` in the update process contributes the most variance among different spatial-based models [21].

2.3.1 Graph Attention Networks

Many GNN models assume the contributions of the neighbouring nodes on the central node's representation are either identical or pre-determined (*e.g.*, [19]) in the aggregation process. In Graph Attention Network (GAT) [20], however, a learned attention layer is used to output a representation based on the weighted average of neighbours' representations. In addition, this method allows us to consider edge features $h_{uv} \in \mathbb{R}^{F_e}$ by including them in the input of the attention layer.

GATv2 [21] has a simple adjustment to the way attention is calculated in GAT, allowing it to provide a more expressive attention layer that is dependent on the query node as opposed to GAT's static attention layer. The attention mechanism which is defined by scoring function $e : \mathbb{R}^F \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ in both of these methods calculates the relative



$\overrightarrow{h_2}$
$\overrightarrow{\alpha}_{12}$
$\overrightarrow{\alpha}_{11}$
$\overrightarrow{h_3}$
$\overrightarrow{\alpha}_{13}$
$\overrightarrow{h_1}$
concat/average
$\overrightarrow{h'_1}$
$\overrightarrow{\alpha}_{14}$
$\overrightarrow{\alpha}_{15}$
$\overrightarrow{h_4}$
$\overrightarrow{h_5}$

Figure 2.3: GAT layer

importance of the features of the neighbor u to the node v as:

$$\begin{aligned} \text{GAT} : e(h_v, h_u) &= \text{LeakyReLU}\left(a^\top . \left[\boldsymbol{W} h_v \| \boldsymbol{W} h_u \| \boldsymbol{W}_e h_{vu}\right]\right), \\ \text{GATv2} : e(h_v, h_u) &= a^\top \text{LeakyReLU}\left(\boldsymbol{W} . \left[h_v \| h_u \| h_{vu}\right]\right), \end{aligned} \tag{2.8}$$

where $\boldsymbol{a}$, $\boldsymbol{W}$, and $\boldsymbol{W}_e$ are learned and $\|$ is the concatenation operator. Using Softmax function, attention scores are then normalized across all neighbours and are used to calculate a new representation for each node by a weighted average (*cf.*, Fig. 2.3) followed by a nonlinearity activation layer (σ):

$$h'_v = \sigma\left(\sum_{u \in \mathcal{N}_v} \text{softmax}_{\mathcal{N}_v}\left(e(h_v, h_u)\right) . \boldsymbol{W} h_u\right). \tag{2.9}$$

Note that once the parameters of attention mechanism and linear transformation, *i.e.*, $\boldsymbol{a}$ and $\boldsymbol{W}$, are learned, alleviating the need for re-training with each topological variation and only (2.8) and (2.9) should be recalculated for affected nodes. In this thesis, we use GATv2 because it has shown to be theoretically and empirically superior to GAT [21].

2.4 Related Work

There are numerous works in the literature that address AC and network slicing [24, 30]. In this section, we review these works with an emphasis on ML-based approaches.

2.4.1 Admission Control

The authors in [31, 16] focus on AC with the objective of maximizing long-term InP revenue. Dandachi et al. [31] propose a traditional RL-based approach for 5G slice admission and congestion control. Even though they consider a slice as a set of VNFs, the substrate network is only considered in aggregate. That is, instead of modeling the substrate network as a collection of interconnected sites or nodes, each with its own limited resources, the network is modeled as a single node with a certain amount of resources. This simplifies the slice embedding problem to an unrealistic degree. Van Huynh et al. [16] leverage DRL for slice AC and resource allocation, but similar to [31], they model slices and the substrate network in aggregate and do not address the RAN slicing problem in terms of multidimensional resource allocation and transport network topology. In addition, both of these works do not consider the dynamic nature of request arrivals.

Pujol Roig et al. [32] propose a DRL-based approach for dynamic VNF management and orchestration. Requests arrive for a list of individual NFs and are embedded on a pool of homogeneous servers in the CU or in the remote cloud. Instead of a binary admission decision, when a new request arrives, a DRL agent decides to either scale the corresponding VNF vertically by allocating more resources to it, instantiate a new VNF on a separate server, or offload the VNF to the cloud. Their objective is to minimize the incurred resource and latency costs. Although the authors deal with the VNF request in a dynamic manner, their model only caters to individual VNFs instead of a network slice. Bega et al. [33] propose an RL-based slice admission solution for maximizing InP revenue. They employ two separate RL agents for estimating revenue in the case of accepting and rejecting SRs, respectively. The authors extended their work in [34] using DRL. However, these works only consider radio resources and require knowledge of the arrival process. **?**] propose an online network slice brokering solution to maximize multiplexing gains. The problem is modeled as a budgeted lock-up multi-armed bandit problem, a variation of the well-known multi-armed bandit problem. Nevertheless, similar to [32, 33, 34], the authors model a network slice as only requiring a number of Physical Resource Blocks (PRBs), whereas a RAN slice consists of a number of functions each with its own latency, computing, and communication resource requirements.

?] propose a policy-based RL algorithm for slice AC in 5G C-RAN. However, the arriving SRs in their work, already specify the required computing resources at the remote and central sites based on the latency requirement. This sidesteps an important aspect of slicing, where all of an SR's functions can be placed at either the remote (*e.g.*, for URLLC applications) or the centralized location (*e.g.*, for mMTC applications). Additionally, the selection of the central location (*i.e.*, remote data center) is done using a heuristic after the AC decision has been made. This precludes the AC agent from knowing the embedding before making the admission decision and can lead to performance degradation in resource-constrained environments.

2.4.2 Slicing

The works discussed in this section assume that SRs will be accepted until resources are saturated. Therefore, online proactive AC is not factored into the problem, and the focus is on optimizing the efficiency (*e.g.*, delay, resource cost, utilization) of resource allocation. Koo et al. [17] leverage DRL for network slicing when requests are served immediately or in batch mode. They consider multi-dimensional resource allocation (*e.g.*, VMs, bandwidth, memory) with delay requirement that includes the processing delay of SRs. However, the authors consider a slice in aggregate.

In contrast, Solozabal et al. [18] use Neural Combinatorial Optimization paradigm for delay-aware service function chain placement. The authors incorporate resource capacity and delay constraints into the objective using Lagrange relaxation. They employ a DRL model architecture which incorporates an encoder-decoder design based on stacked Long Short-term Memory cells. The model can decide the placement for the whole chain of VNFs. However, to simplify the path selection, the servers are assumed to be connected through a star topology.

Yu et al. [35] were the first to investigate the 3-layer RAN slicing in the context of metro networks. They analyzed the problem of CU/DU placement and routing to minimize the number of central offices (COs) housing the functions under fronthaul delay and capacity constraints. The authors showed that the increased flexibility of a 3-layer RAN architecture leads to a higher consolidation of COs. Based on the same architecture, Xiao et al. [36] proposed a MILP and a heuristic to optimize energy efficiency by modelling the power consumption of different components of the network, while Yu et al. [37] investigated isolation-aware slicing and proposed a heuristic for minimizing the number of active COs or wavelengths under isolation and latency constraints. Marotta et al. [38] addressed the same problem but also took into account the reliability requirements of different slices.

The basic idea behind the heuristic methods adopted in these works is to first place the functions using analytical modelling and heuristic methods and then route using variants of the shortest-path algorithm. In addition, they work in an offline setting, *i.e.*, an objective function is optimized over all the requests.

Gao et al. [14] developed a DRL-based method for online RAN function placement and routing from RU to the data center with the objective of minimizing the number of active COs, bandwidth, and transport latency. However, their evaluation is limited to a single service (*i.e.*, slice) type and they do not consider different E2E service latency constraints and slices with finite operation time. The works in [11, 12, 13] investigated function placement in the context of MEC-enabled RAN. They modeled the problem of minimizing the operational cost under delay and capacity constraints as an ILP and solved it using Benders Decomposition [11, 12] and DRL [13]. However, in these methods, the problem is considered in an offline setting and placement is decided for each cell instead of each request.

Also related are works that consider the functional splitting problem in which the placements of RAN functions are optimized on an interconnected set of DU and CU servers [39, 40]. The authors in [41, 39] address the user-centric functional split problem, where function placement decisions are made for each request. They model the problem as an ILP, and propose solutions based on particle swarm optimization and deep learning, respectively. However, the authors model the substrate network as only having a single RU and a single CU. Wang and Zhang [40] consider a pool of DU and CU servers connected hierarchically and maximize profit (*i.e.*, the difference between revenue and cost) by using traditional RL. However, to simplify the problem, the authors divide it into function embedding and radio resource allocation, and solve them individually (*i.e.*, using different Q-learning models) rather than jointly. Sulaiman et al. [1] proposed an online joint RAN slicing and AC solution under E2E service delay and resource constraints using multi-agent DRL. They showed that the AC mechanism can lead to a higher revenue by preemptively rejecting low priority slice requests. However, their method is not scalable and generalizable to large and previously unseen substrate networks.

2.4.3 GNN-based Architectures

Recently, DL-based approaches using GNNs have been incorporated to address the problems of scalability and generalizability when working with graph-based network topologies. The problem of RAN slicing shares many similarities with the VNE problem. VNE is a resource allocation problem which involves mapping a virtual network to the substrate network, and is highly studied using heuristic and mathematical modelling. However, these methods are either inefficient or impractical when the substrate network is a large-scale graph. [24]. GNN-based architectures have been employed in networking for applications such as VNE [42], traffic routing [43], and congestion prediction and interpretability [44], to capture the spatial information hidden in the network topology [42] and benefit from its generalizability to different topologies [43, 44]. Yan et al. [42] were the first to apply DRL with GNNs, specifically GCN, to the VNE problem and showed that it can lead to a higher acceptance ratio, performance and robustness. They also use a multi-objective reward function with parallel training to improve convergence and performance. The DRL agent places the VNFs one-by-one on substrate nodes with sufficient CPU and bandwidth, and then, the shortest-path algorithm is utilized for routing between the selected nodes. However, their approach does not consider E2E delay constraints and utilizes MLPs, which limits its applicability to previously unseen networks.

Zhang et al. [45] investigated the same problem when VNs can dynamically change over time. However, GNN-based DRL is solely used for the initial mapping and a heuristic is leveraged to remap the VN when a change happens. Esteves et al. [46] extended the approach in [42] by considering more resource types for each VNF and using heuristically assisted DRL for faster convergence. However, the use of spectral-based GNN models (*cf.*, Section 2.3.1) and dense layers in the learning model limit the applicability of these works to static network topologies. Habibi et al. [47] employed a spatial-based graph autoencoder to cluster similar substrate nodes based on their resources and accessibility. However, the final node and link embeddings across clusters are decided using a Breadth-First Search (BFS) algorithm which is not efficient.

Chapter 3

Design

3.1 RAN and MEC Architecture

One of the main changes of the 5G NR transport network is the support for RAN functional split. 3GPP proposed that the BBU functions in 4G/LTE can be decomposed in 5G NR into three entities, RU, DU, and CU [3]. Fig. 3.1 outlines RAN functions in physical (RF, PHY) and data link (MAC, RLC, PDCP) layers [48] and different split options proposed by 3GPP [3]. This flexible and disaggregated RAN architecture facilitates RAN slicing and enables network customization as per the needs of slice tenants. Moving from lower to higher layer splits increases multiplexing and centralization gains at the cost of increased bandwidth and higher delay.

Aligned with 3GPP and ITU-T recommendation [4], in this thesis, we consider options 2 and 7 for higher layer (CU/DU) and lower layer (DU/RU) splits, respectively, and use the terms midhaul-I (MH-I) and fronthaul (FH) to refer to the corresponding transport network segments. In addition, we assume, based on the control-plane/user-plane split, that the CU includes only the user-plane PDCP function and control-plane functions are placed in the core network (CN). This configuration is particularly useful for applications that do not require rapid call establishment but require a low user-plane delay, *e.g.*, cloud gaming [8]. MEC can be integrated with the 5G network in various ways [5]. We assume that the MEC includes edge application, as well as local user-plane functions (UPF) and is connected through the midhaul-II (MH-II) and backhaul (BH) network to the CU and CN, respectively. Finally, CN is connected to the Internet and provides access to other parts of the network.

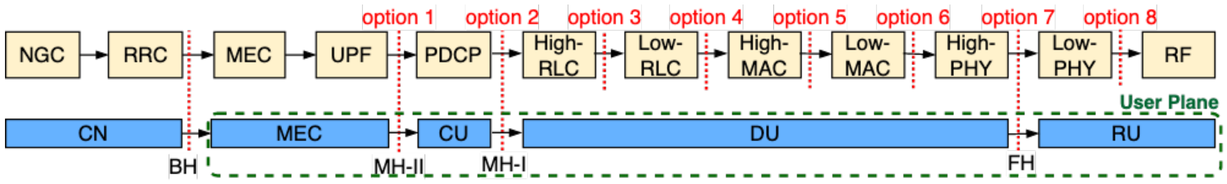


Figure 3.1: Mapping RU, DU, CU, and MEC functions and FH, MH-I, MH-II, and BH networks to the split points [3, 4, 5]

For each network slice, the individual RU, DU, CU, and MEC entities are virtualized and placed at different physical locations according to their latency and resource requirements, and residual capacity in the substrate network. We assume RU is placed on the access node connected directly to the originating cell and CN is placed on the core node, which has abundant capacity, and therefore, we only decide the placement of DU, CU, and MEC, and routing from the access site to the core node. The specifications of MEC depend on the type of service, but we use the following formulations to calculate the computation ($c^{\text{cpu}}_{\text{DU}}$ and $c^{\text{cpu}}_{\text{CU}}$ in Giga Operations Per Second (GOPS)) and bandwidth (λ_l in Mbps) requirements of RAN components and their interconnecting transport segments, respectively.

$$c^{\text{cpu}}_{\text{RU}} = k^{\text{RU}}_1 BA + k^{\text{RU}}_2 BAL, \tag{3.1}$$
$$c^{\text{cpu}}_{\text{DU}} = k^{\text{DU}}_1 BA^2L + k^{\text{DU}}_2 BALM + k^{\text{DU}}_3 A, \tag{3.2}$$
$$c^{\text{cpu}}_{\text{CU}} = k^{\text{CU}}_1 A, \tag{3.3}$$
$$\lambda_l = k^l_1 \lambda^{\text{new}} + k^l_2,\ \forall l \in \{\text{FH}, \text{MH-I}, \text{MH-II}, \text{BH}\}, \tag{3.4}$$

where k parameters in (3.1)-(3.4) are constant coefficients specific to different RAN functions whose details can be found in [49, 50] and [51], respectively. λ^{new}, B, A, L, and M represent the service traffic, carrier bandwidth, number of antennas, traffic load, and modulation (in bits per symbol). While MH-I (*i.e.*, $\lambda_{\text{MH-I}}$) and MH-II (*i.e.*, $\lambda_{\text{MH-II}}$) bandwidth requirements for each slice request is almost equal to λ^{new}, FH interface (*i.e.*, λ_{FH}) requires considerably higher bandwidth. The bandwidth requirement at the BH network (*i.e.*, λ_{BH}) depends on the amount of Internet traffic of each type of service. Moreover, each RAN component has a specific delay requirement. From the RAN perspective, for DU, depending on the specific vendor implementation of the HARQ loop, it can be up to 2 ms for the case where interleaving is done, and for CU, it could be up to 6 ms [50]. However, since DU and CU are before MEC in the user-plane, their delay requirements depend on both the respective RAN component and the E2E service latency, *i.e.*, the minimum value

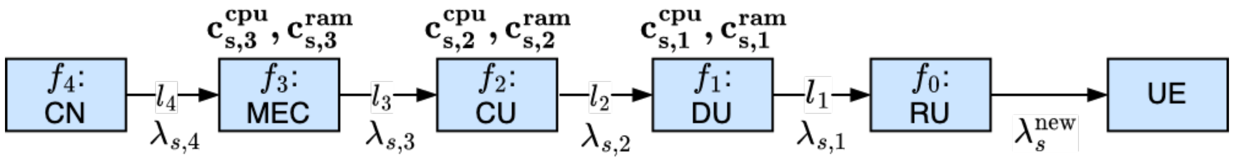


Figure 3.2: Virtual network model of service s

should be considered.

3.2 System Model

Substrate Network: We consider a network architecture consisting of $\mathcal{N}$ access and aggregation sites equipped with dedicated processing capabilities. Access nodes can be connected to one or multiple cell sites. We also consider a core[1] node with abundant resources which can host the CN. These nodes are connected through an undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the union of $\mathcal{N}$ and core node (index 0), and $\mathcal{E}$ is the set of all physical links. We denote by C_v^{cpu} and C_v^{ram} the maximum computing and RAM resource capacities of each node $v \in \mathcal{V}$, respectively. Also, each link $e \in \mathcal{E}$ has a certain bandwidth capacity, B_e, and delay, d_e.

Services: We consider a set of $\mathcal{S}$ services, *i.e.*, MEC applications. Throughput and E2E service delay constitute QoS metrics, and are denoted by λ_s^{new} and D_s^{srv}, respectively. RAN and MEC form a VN consisting of a chain of five VNFs, $\mathcal{F} = \{f_0, \cdots, f_4\}$, namely, RU, DU, CU, MEC, and CN, and four VLs $\mathcal{L} = \{l_1, \cdots, l_4\}$, namely, FH, MH-I, MH-II, and BH (*cf.*, Fig. 3.2). For each service s, we can denote the computing and memory resource requirements of VNF f, and bandwidth requirement on VL l by $c_{s,f}^{\text{cpu}}$, $c_{s,f}^{\text{ram}}$, and $\lambda_{s,l}$, respectively. The CPU and bandwidth requirements can be calculated according to equations (3.2)-(3.4) for RAN VNFs. We assume that RAM requirements follow the same pattern as computation requirements for RAN VNFs. For MEC, these requirements depend on the computational complexity of the specific application. The delay requirement of VNF f is also shown by $D_{s,f}$, which for MEC is equal to D_s^{srv}.

Slice Requests (SRs): Requests arrive for different services over time. We characterize each generic SR k by its service type, the access node on which this request was first submitted, n_k^{src}, offered revenue, p_k, and the set of time slots it needs to receive service

[1]Formulation can trivially be extended to networks with more than one core node.

($i.e.$, operation time), $\mathcal{T}_k = \{t_k^{\text{arv}}, \cdots, t_k^{\text{arv}} + \tau_k - 1\}$. We analyze the system over period $\mathcal{T}$ and denote the set of all SRs for service s and for all the services that arrive over this period by $\mathcal{K}_s$ and $\mathcal{K}$, respectively.

3.3 Problem formulation

Given the knowledge of future SRs, the offline RAN slicing and AC problem should decide about the admission of each SR and the embedding of its corresponding VN, *i.e.*, placing VNFs over physical nodes and routing traffic between them through physical links.

Decision variables: Let $\alpha = [\alpha_k]_{k\in\mathcal{K}}$ denote admission matrix, where $\alpha_k \in \{0,1\}$ denote whether SR k is admitted ($\alpha_k = 1$) or not ($\alpha_k = 0$). VNF embedding decisions are defined by the matrix $\mathbf{X} = [x_{f,v}^k]_{r\in\mathcal{R},f\in\mathcal{F},v\in\mathcal{V}}$, where $x_{f,v}^k \in \{0,1\}$ indicate whether VNF f of SR k has been placed on physical node v ($x_{f,v}^k = 1$) or not ($x_{f,v}^k = 0$). In this work, we consider single-path routing, *i.e.*, each VL is mapped to one physical path. Let $q \in \mathcal{Q}$ denote a simple path in the physical network. A path is a sequence of links between two nodes $src(q), dst(q) \in \mathcal{V}$. We assume $\emptyset \in \mathcal{Q}$ for which $src(q) = dst(q)$, to consider co-location of VNFs. So, we define matrix $\mathbf{Y} = [y_{l,q}^k]_{k\in\mathcal{K},l\in\mathcal{L},q\in\mathcal{Q}}$ to describe VL embedding decisions, where $y_{l,q}^k \in \{0,1\}$ indicates whether the traffic of VL l in SR k has passed through q ($y_{l,q}^k = 1$) or not ($y_{l,q}^k = 0$).

Admission and embedding constraint: If SR k is admitted, it will remain in the system throughout its operation time, $t \in \mathcal{T}_k$, and the corresponding VN should be embedded into the substrate network. The following constraint ensures the embedding of each VNF and VL of an SR on one physical node and one physical path, respectively, if and only if it is admitted:

$$\alpha_k = \sum_{v\in\mathcal{V}_{k,f}} x_{f,v}^k = \sum_{q\in\mathcal{Q}} y_{l,q}^k,\ \forall k, f, \tag{3.5}$$

where $\mathcal{V}_{k,f} \subseteq \mathcal{V}$ includes the physical nodes on which VNF f of SR k can be placed. As discussed, the placement is decided for DU, CU, and MEC for which $\mathcal{V}_{k,1} = \mathcal{V}_{k,2} = \mathcal{V}_{k,3} = \mathcal{V}, \forall k$. However, for the sake of formulation, we also define VNF embedding variables for CN and RU and set $\mathcal{V}_{k,4} = \{0\}$ and $\mathcal{V}_{k,0} = \{n_k^{\text{src}}\}$ to impose the placement of CN and RU on the core node and originating access site, respectively.

Routing constraints: The problem of selecting paths for embedding each VL can be framed as the well-known *unsplittable multi-commodity flow* problem [52]. Therefore, routing variables specified by VL embedding matrix, $\mathbf{Y}$, should meet the below flow conservation constraint, where $src(l), dst(l) \in \mathcal{F}$ are source and destination VNFs of VL l:

$$\sum_{\substack{q\in\mathcal{Q}:\\ src(q)=v}} y^k_{l,q} - \sum_{\substack{q\in\mathcal{Q}:\\ dst(q)=v}} y^k_{l,q} = x^k_{src(l),v} - x^k_{dst(l),v},\ \forall k, v, l. \tag{3.6}$$

Capacity constraints: The assigned resources to VNFs and VLs should not exceed the capacity of nodes and links in the substrate network. These constraints are expressed as:

$$\sum_{s\in\mathcal{S}} \sum_{\substack{k\in\mathcal{K}_s\\ :t\in\mathcal{T}_k}} \sum_{f\in\mathcal{F}} c^{\text{cpu}}_{s,f} x^k_{f,v} \le C^{\text{cpu}}_v,\ \forall v, t\in\mathcal{T}, \tag{3.7}$$

$$\sum_{s\in\mathcal{S}} \sum_{\substack{k\in\mathcal{K}_s\\ :t\in\mathcal{T}_k}} \sum_{f\in\mathcal{F}} c^{\text{ram}}_{s,f} x^k_{f,v} \le C^{\text{ram}}_v,\ \forall v, t\in\mathcal{T}, \tag{3.8}$$

$$\sum_{s\in\mathcal{S}} \sum_{\substack{k\in\mathcal{K}_s\\ :t\in\mathcal{T}_k}} \sum_{l\in\mathcal{L}} \sum_{\substack{q\in\mathcal{Q}\\ :e\in q}} \lambda_{s,l} y^k_{l,q} \le B_e,\ \forall e, t\in\mathcal{T}. \tag{3.9}$$

In the above expressions, the total used resources at time t are computed as the sum of resource requirements of active embedded SRs at that time, *i.e.*, each SR k for which $t \in \mathcal{T}_k$.

Delay constraints: Finally, E2E service delay and individual VNF latency constraints for each SR are expressed as:

$$\sum_{\substack{l\in\mathcal{L}:\\ l\le f}} \sum_{q\in\mathcal{Q}} y^k_{l,q} \sum_{e\in q} d_e \le D_{s,f},\ \forall s, k\in\mathcal{K}_s, 1\le f\le 3. \tag{3.10}$$

Objective: The goal of the InP is to grant SRs that lead to the highest long-term revenue. We can formulate the problem of RAN slicing and AC while optimizing the revenue of InP over all the requests (or period $\mathcal{T}$) as

$$\max_{\alpha,\mathbf{X},\mathbf{Y}} \sum_{k\in\mathcal{K}} p_k \tau_k \alpha_k \quad \text{subject to} \quad (3.5) - (3.10).$$

The above problem has linear constraints and includes integer variables, and hence it is an ILP problem. This problem is NP-hard in the offline setting since if delay constraints are set large enough to eliminate (3.10), it will become the VNE problem which is NP-hard [53]. In the next chapter, we propose a DRL-based approach to tackle this problem in the online setting, where SRs information is not available beforehand.

3.4 Challenges

3.4.1 Large Problem Space

The primary challenge of this work is with regards to the scope of the problem and the current capacity of DRL. In DRL, the problem space refers to the set of all possible states, actions, and transitions that the agent can encounter as it interacts with its environment. A large problem space can pose several challenges for an agent trying to learn an optimal policy. With the increase in substrate network size, the agent must process sizable information to make viable decisions. Additionally, the number of possible slice embedding decisions increases exponentially with network size. Such circumstances reside in the limits of what DRL can successfully tackle and thus, require the use of more sophisticated learning algorithms and more powerful computing resources to train complex DNNs that are more difficult to train [54]. Therefore, it is crucial to optimize the training scenario for the algorithm by designing a reward function that leads to the optimal policy, while also reducing the action space by breaking down the problem into multiple stages.

In this work, we simplify the problem by breaking down the slicing action into multiple steps, each pertaining to a single VNF placement. Moreover, we design a model architecture that divides the large observation/action space into node-wise values, as opposed to dealing with massive flattened vectors. This will be explained further in Chapter 4.

3.4.2 Low Action Temporal Correlation

Since slice requests arrive randomly in different locations of the substrate network and delay constraints limit the valid VNF placement decisions to a relatively small neighborhood of nodes, the impacts of a certain slice placement decision do not affect the decisions of the agent until far into the future. To elaborate further, for instance, if a slice embedding creates a bottleneck, the section of the network where this bottleneck is located will be affected. However, due to the large number of physical nodes, it might take several new requests until a new slice request arrives in the bottleneck's vicinity.

This is not in-line with an optimal DRL scenario, where as previously explained (*cf.*, 2.2.1), the majority of algorithms operate by comparing the effects of an action (*i.e.*, the received reward) with the expected reward. If the received reward is higher than anticipated, the agent learns to increase the probability of selecting that action, and vice versa. The discount factor γ dictates how far into the future the agent considers when analyzing the impacts of its decisions, by calculating the discounted sum of rewards at each step. If this parameter is set too large, the differences between the actual and the expected sum of rewards become unnoticeable, leaving no margin of error and hindering the agent's training process. Alternatively, if this value is set too small, the impacts of each action will be ignored. Furthermore, this challenge is intensified when each of the slicing decisions is spread into multiple steps, furthering reward sparsity. To solve this issue, we opted to train the models on a smaller topology, which reduced the sparsity of the action impacts. However, the trained models are still able to maintain their performance on larger topologies, as evident by the results (*cf.*, Fig. 5.15).

3.4.3 Limited Valid Actions

As previously mentioned, with large substrate networks and constrained end-to-end delays, valid actions are limited to a relatively small portion of the overall available physical nodes. The larger the network or the more restrictive the delay constraints, the proportion of the valid to invalid placements becomes smaller. Since the DNN model starts with randomized weights, the DRL agent's initial decisions are totally random. If the aforementioned proportion becomes too low, the agent will get trapped in a loop of invalid actions, collecting poor trajectories that will be used to train the model. This severely degrades the training performance. Conventional action selection methods deal with these invalid actions by issuing a large negative reward as a disincentive. However, with the ratio of valid to invalid actions in our case, such technique would hinder the agent's ability to learn and converge on an optimal policy.

Training on a smaller topology did not mandate employing a technique to handle this issue. However, when testing on larger topologies, we utilize action masking [55] which enables the slicing agent to explore in a smarter way by masking out invalid actions at any given state. For discrete action spaces, the actor outputs a logit per action. These logits are used to calculate an action selection probability distribution using the Softmax function. In action masking, invalid actions' logits are manually adjusted to $-\infty$, which translates to a probability of zero in the Softmax function. This way, we force the agent to select from the remaining valid actions.

3.4.4 Complexity of the Optimal Policy

Reward design is an important aspect of designing any RL-based solution. As equations (2.5)-(2.6) suggest, when taking an action, a DRL agent not only takes the immediate reward into account but also the discounted rewards it expects to receive in the future. Consequently, the agent is able to learn meaningful state-action values for the state-action pairs for which it receives no immediate reward. The more fine-grained the rewarding function is, the higher is the chance of a successful DRL application. A detailed rewarding function gradually guides the agent towards the optimal policy. In contrast, the sparsity of the reward greatly affects the convergence of a DRL agent's policy, as it would have to spend more training steps to find the path to the optimal policy. This makes learning from sparse rewards one of the major challenges in DRL [56]. Reward shaping refers to providing the agent with additional carefully designed rewards, such that they guide the agent towards the desired behavior faster. Careless reward shaping can often lead to unintended behavior [57]. For example, if a self-driving car is only given a positive reward each time it moves towards the final destination, it may learn to move in circles without ever actually reaching the final destination.

Nevertheless, finding sub-objectives that can lead to the optimal policy and can be translated into specific rewards is not always trivial. In the problem of RAN slicing in metropolitan 5G networks, the immensity of the problem in both temporal and spatial aspects, makes hand-tuned naïve policies inadequate for framing the rewarding function. For instance, positive reinforcement for placement of the virtual functions closer to the core (*i.e.*, a centralized approach) is not guaranteed to be in-line with the optimal policy. Alternatively, utilizing an ILP method to extract the optimal solution is proven to be computationally infeasible for large topologies. Therefore, we employed a reward function that involves just the achieved revenue for the AC agent and the total deployed slice requests for the Slicing agent, to ensure that the agent only moves towards increasing the total achieved revenue.

3.4.5 Generalizable Model Architecture

The main novelty of this work is regarding its ability to accommodate varying substrate network topologies, without the need to re-tune or re-train the trained model. To accomplish this, our DNN model would have to be indifferent to the topology. Fully connected neural network layers are most commonly used in artificial neural networks, however, these layers are reliant on fixed input/output dimensions and ordering. A novel design is required, such that it is not only able to operate with different sized and ordering input/output, but that it also maintains a general understanding of the graph structures. GNNs are most suitable for this task, however, they are notoriously difficult to train [58]. As a result, they are commonly combined with additional fully connected layers, improving their performance at the cost of eliminating their generalizability [42, 45, 46]. However, in this work, we refrained from doing so and employed a novel architecture that works with arbitrary substrate networks. The architecture is explained in details in Chapter 4.

Chapter 4

Solution

4.1 GNN-based Multi-agent DRL Framework

The online scenario requires that SRs be handled one-by-one as they arrive, without being aware of future SRs, such that the total revenue of the InP is maximized over the long term. When a request arrives, the online DRL-based solution should decide on its admission and VN embedding based on the current network state. To maximize InP revenue, the agent should be rewarded for successful embeddings and revenue generation. However, such reward design may lead to unintended behaviour, as it is not clear whether the lost revenues are a consequence of suboptimal previous admission or embedding actions. To tackle this issue, similar to [1], we use two agents, slicing and AC, that operate in a coordinated manner.

In our proposed solution, called GNN-AC-SL, first, the slicing agent optimizes the embedding based on the SR's specifications and network state to maximize the number of embedded requests. The AC agent then decides whether to accept the SR based on the embedding decision, the system state, and the SR's information, so that long-term revenue is maximized. Even though decoupling AC and slicing reduces the action space, the slicing agent still has to select a chain in the substrate network that has $|\mathcal{V}|^{|\mathcal{F}|}$ possibilities. We reduce the action space of the slicing agent by transforming VN embedding into a sequence of VNF embeddings. As a result, the slicing agent must only choose between the physical nodes at each time. This section will provide a detailed explanation of the two DRL-based agents.

4.1.1 RL Environment

As previously mentioned, each RL solution includes a state (observation) space definition, an action space definition, and a reward function. In this section, we will explain how we converted the slicing and admission control problems into DRL scenarios.

State (s_t): For the slicing agent, we represent the state of the whole system as a graph, *i.e.*, $s_t = (\mathcal{V}, \mathcal{E})$, with node and edge attributes. Since VNFs will be placed one-by-one, the current VNF should also be considered in the state of the system. We define the node features by $\mathbf{X}^{\mathrm{sl}} = [X_v^{\mathrm{sl}}]_{v \in \mathcal{V}}$, where $X_v^{\mathrm{sl}} \in \mathbb{R}^{F^{\mathrm{sl}}}$ is the feature vector of node v and consists of its maximum and remaining CPU and RAM, its tier in the substrate network (*i.e.*, access, aggregate or core), and the delay and remaining bandwidth from the last placed VNF. Moreover, X_v^{sl} includes features of the current SR and its VNFs, namely the operation time, the index and delay budget of the current VNF, and the required CPU and RAM by the last and the current VNFs.

Similarly, the matrix $\mathbf{X}^{\mathrm{sl,e}} = [X_{vu}^{\mathrm{sl,e}}]_{(v,u) \in \mathcal{E}}$ represents the edge features of the whole graph, where $X_{vu}^{\mathrm{sl,e}} \in \mathbb{R}^{F^{\mathrm{sl,e}}}$ is the feature vector of the edge between nodes v and u and includes the maximum and remaining bandwidth capacity, and the link delay.

Different from the slicing agent, the AC agent only considers the hosting nodes and edges of the substrate network based on the given embedding decided by the slicing agent, and incorporates the amount of required resources from them into their features. The input feature of the AC agent is an F^{ac}-dimensional vector $\mathbf{X}^{\mathrm{ac}} \in \mathbb{R}^{F^{\mathrm{ac}}}$ which includes the hosting nodes' maximum and remaining CPU and RAM, the tier in the substrate network, and the requested CPU and RAM by the embedding, along with the embedding's sum of requested bandwidth across the substrate network, the SR's operation time, and the SR's revenue.

Action (a_t): There are only two possible actions for the AC agent: accepting or rejecting the current slice request, *i.e.*, AC's action space is equal to $\mathcal{A}^{\mathrm{ac}} = \{0, 1\}$. The slicing agent, however, has to select a chain in the substrate network that has $|\mathcal{V}|^{|\mathcal{F}|}$ possibilities. We reduce the action space of the slicing agent by transforming VN embedding into a sequence of VNF embeddings. As a result, the slicing agent must only choose between eligible physical nodes at each time. For each SR, we embed four VNFs in a sequence and VL associated with each VNF is embedded using a shortest-path algorithm. Thus, for the slicing agent, the set of all possible actions is limited to the number of substrate nodes, *i.e.*, $\mathcal{A}^{\mathrm{sl}} = \mathcal{V}$.

Reward (r_t): A total reward of +1 is given to the slicing agent for the successful embedding of all four RAN functions, and partial embedding is not rewarded in order

Algorithm 1: Slicing and Admission Control

```
Input: action $a_t$, current agent $\delta_{curr}$
Output: next state $s_{t+1}$, reward $r_t$, next agent $\delta_{next}$
$r_t \leftarrow 0$
if $\delta_{curr}$ ==“Slicing” then
    if checkActionFeasibility($a_t$) then
        embedding.add($a_t$)
        if len(embedding) == 4 then                      // SR embedding is complete
            $r_t \leftarrow 1$
            $\delta_{next}$ ←“Admission Control”
        else
            moveToNextVNF()
            $\delta_{next}$ ← “Slicing”
    else
        moveToNextSliceRequest()                         // infeasible SR
        embedding ← $\emptyset$
        $\delta_{next}$ ←“Slicing”
else if $\delta_{curr}$ ==“Admission Control” then
    if $a_t$ == 1 then                                   // SR is admitted for deployment
        deploy(embedding)
        $r_t$ ← revenue
    $\delta_{next}$ ←“Slicing”, go to Line 12
$s_{t+1}$ ← readState()
return $s_{t+1}, r_t, \delta_{next}$
```

for the agent to learn to embed the requests in their complete form. Using this system, a rewarding decision includes up to four steps until an SR embedding succeeds or fails. Likewise, the AC agent receives +1 reward for admission of each SR.

Alg. 1 outlines the simulated RL environment algorithm. At each timestep, the environment receives the action and the id of the agent which issued it. In case the action was issued by the slicing agent, the environment checks the feasibility of the action, *i.e.*, whether the VNF placement satisfies all constraints or not. If the action does not violate any SLAs, it is added to the current embedding; otherwise, the environment rejects the current slice request, resets the embedding, and moves on to the arrival of the next slice requests. If every VNF in the SR has been placed, the environment calls the AC agent for the next iteration. If the AC agent decides to deploy the current slice request, the respective resources are deducted from the overall capacity of the network, else, the environment moves on to the arrival of the next slice request.

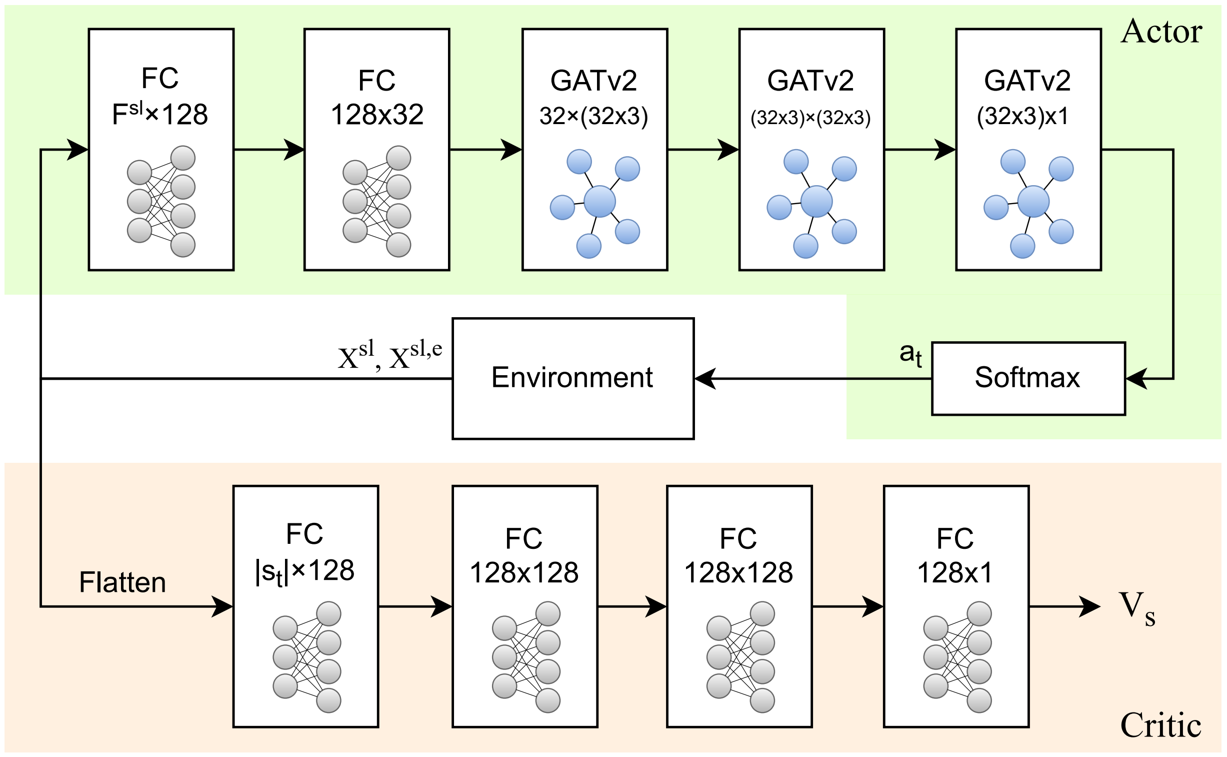


Figure 4.1: Slicing agent model architecture

4.1.2 Training Algorithm

We train both AC and slicing agents together as a multi-agent DRL scenario and utilize the PPO algorithm, as it is one of the leading DRL methods.

The NN architecture of actor and critic modules for the slicing agent is shown in Fig. 4.1. As previously explained, a specific design is required so that the model can support previously unseen topologies, without re-tuning. In the actor, first, the node-wise input is passed through two fully connected (FC) layers to create embeddings that are later used by the GNN layers. Although these layers are FC, they operate on a per-node basis, meaning network information is fed to them as a batch of single-node feature vectors, X_v^{sl}, which has a constant size F^{sl}. Therefore, even if the number or the order of nodes changes, it will not affect the output of the model.

Next, three GATv2 layers are used, each with three attention heads of size 32, that are required in order to share information among neighboring nodes and thus, pass knowledge

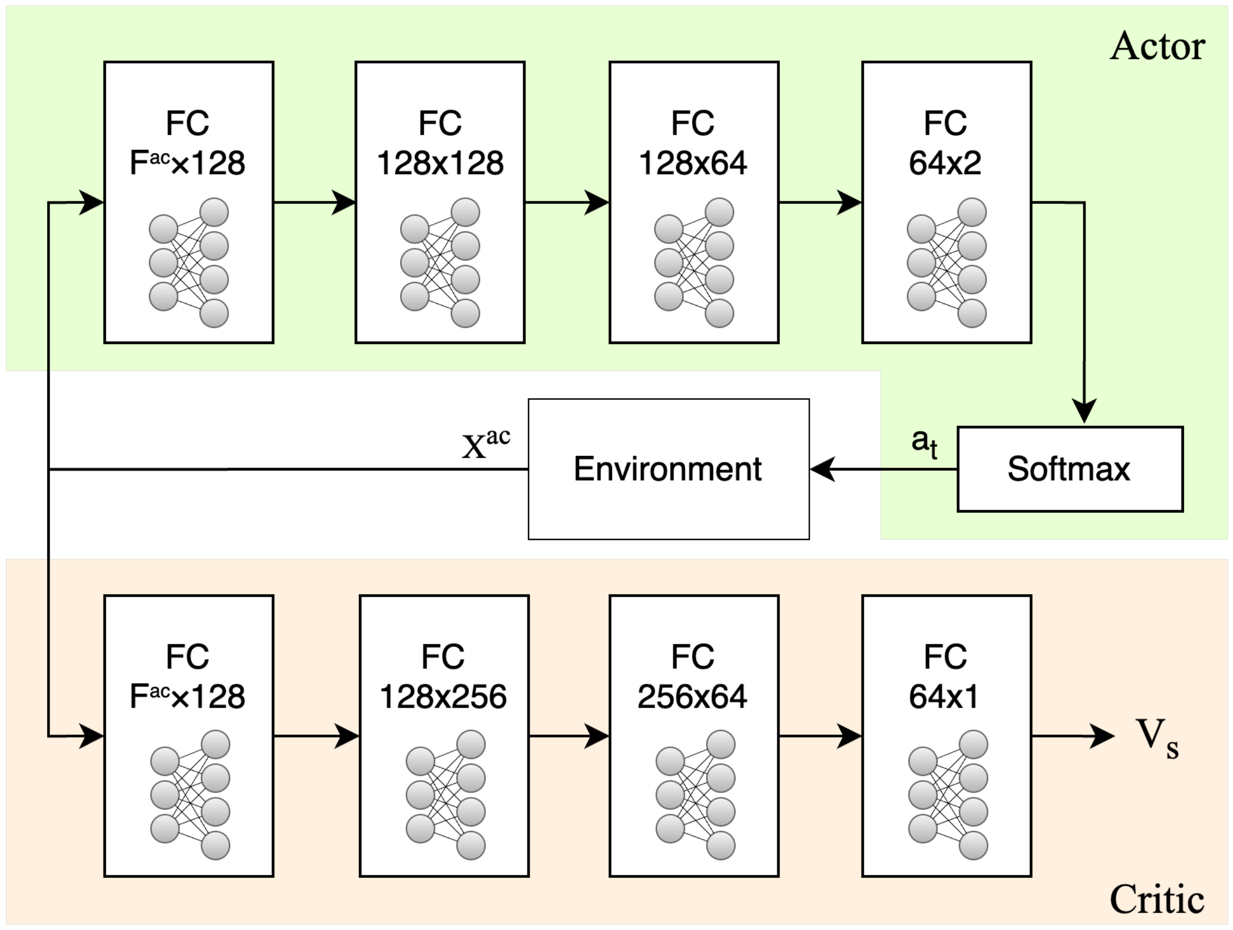


Figure 4.2: Admission control agent model architecture

across the graph. Finally, the model outputs a single value per node, and a Softmax function is used to determine the node-selection probability distribution. Note that the GNN models used in the previous works (*e.g.*, [42]) required embedding the link features into the node by aggregating the features of the edges which directly link to that node. Using GATv2 in this work, we are able to pass the network state as a graph-structured data with features for both nodes and edges.

While designing GNN-based models, a total number of 2 to 4 GNN layers is usually recommended, as they have been found to be effective in practice for many graph-related tasks while not overly increasing the risk of over-smoothing [19]. Over-smoothing occurs when a GNN with a large number of layers propagates information too far through the graph, causing the node representations to become indistinguishable from one another. Through experiments, we found 3 GATv2 layers to achieve the best performance during evaluation. However, 2 additional FC layers were proven necessary due to the complexity of the problem at hand. While these FC layers improve the performance of the model in capturing node information, they do not include the message-passing process that could potentially contribute to the over-smoothing problem.

Due to the simpler objective of the critic model and the fact that it is only used during the training phase in which the topology is fixed, we use an MLP model for it which has a faster convergence time. The model is fed with the flattened $|s_t|$-dimensional representation of the state s_t that is stripped of some repetitive features. In the same vein, as explained in Section 4.1.1, since the input dimensions of the AC agent remain static regardless of the substrate network size and since it has a small action space of size 2, we opt for a less complex MLP-based architecture for the AC agent which consists of four fully connected layers, as shown in Fig. 4.2. This model's actor outputs two values corresponding to acceptance and rejection of the current slice request. A Softmax function is used to generate an action probability distribution, from which the final action is drawn.

Chapter 5

Performance Evaluation

5.1 Simulation Setup

We consider four delay-sensitive MEC applications shown in Table 5.4 with corresponding MEC delay, CPU and RAM usage of the MEC VNF, and backhaul traffic [7]. Parameters of other VNFs and VLs in the VN are derived as discussed in Chapter 3, and displayed in Tables 5.1-5.3 with RU configuration of 20MHz, 4x4 MIMO, and 64QAM. A simulation is run for 2000 time units with random SRs arriving heterogeneously at different access nodes with a total rate of 1 SR per time unit. Each SR has an operation time following a normal random distribution of $\mathcal{N}(300, 25)$ and is uniformly assigned to a MEC application and to a priority class (*i.e.*, high-priority (HP) or low-priority (LP)). Each slice request has a random revenue based on its MEC application and following the distributions in Table 5.4, and a HP SR offers twice the revenue of a LP SR.

Table 5.1: RU characteristics [7]

Application	**RU CPU** (GOPS)	**RU RAM** (GiB)	**RU Latency** (ms)	$\boldsymbol{\lambda}^{\text{new}}$ (Mbps)
Remote Surgery (RS)	1608	32.16	0.25	20
Cloud Gaming (CG)	1640	32.8	0.25	100
Virtual Reality (VR)	1640	32.8	0.25	100
Video Streaming (VS)	1680	33.6	0.25	200

Table 5.2: DU characteristics [7]

Application	**DU CPU** (GOPS)	**DU RAM** (GiB)	**DU Latency** (ms)	**DU Traffic** (Mbps)
Remote Surgery (RS)	312.4	6.25	1	139
Cloud Gaming (CG)	362.2	7.25	2	651
Virtual Reality (VR)	362.2	7.25	2	651
Video Streaming (VS)	424.4	8.5	2	1291

Table 5.3: CU characteristics [7]

Application	**CU CPU** (GOPS)	**CU RAM** (GiB)	**CU Latency** (ms)	**CU Traffic** (Mbps)
Remote Surgery (RS)	100	2	1	20
Cloud Gaming (CG)	100	2	5	100
Virtual Reality (VR)	100	2	6	100
Video Streaming (VS)	100	2	6	200

Table 5.4: MEC applications with characteristics and corresponding revenues [7]

Application	**MEC CPU** (GOPS)	**MEC RAM** (GiB)	**MEC Latency** (ms)	**Backhaul Traffic** (Mbps)	**Revenue**
Remote Surgery (RS)	200	10	1	10	$\mathcal{N}(80, 5)$
Cloud Gaming (CG)	1500	30	5	30	$\mathcal{N}(80, 10)$
Virtual Reality (VR)	2000	60	10	30	$\mathcal{N}(100, 10)$
Video Streaming (VS)	150	10	200	60	$\mathcal{N}(70, 10)$

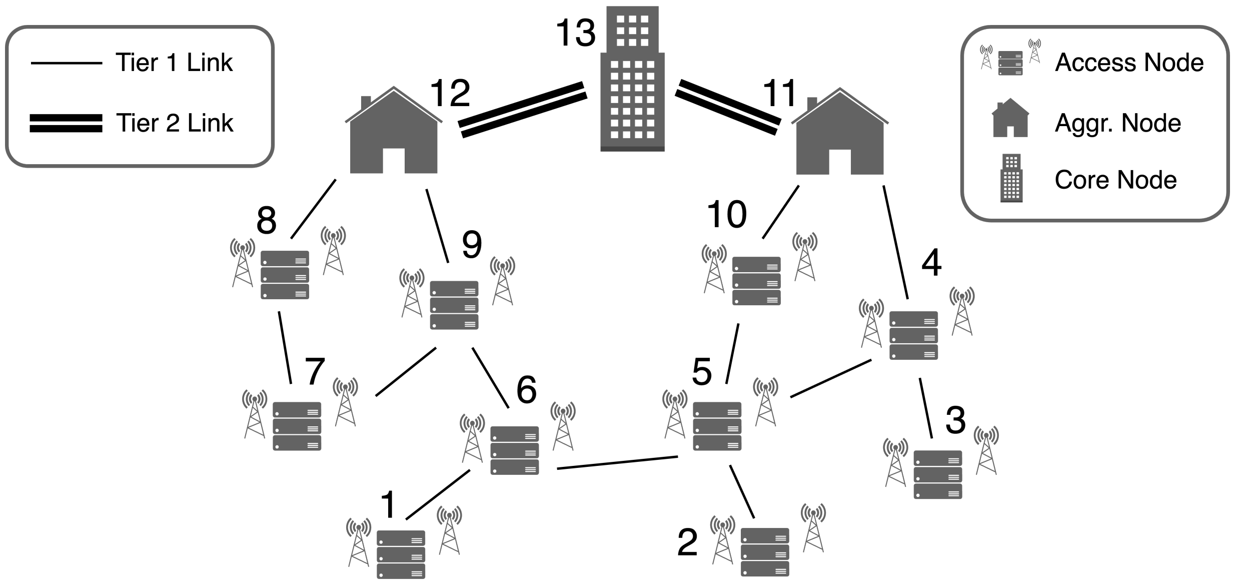


Figure 5.1: Baseline metropolitan 5G network used for training

We train our agents on the topology shown in Fig. 5.1, which comprises of 10 access nodes, 2 aggregation nodes, and 1 core node. CPU capacities are equal to 4000, 6000, and 12000 GOPS, and RAM capacities are 100, 150, and 300 GB, for access, aggregation, and core nodes, respectively. Links connecting access nodes to each other and to aggregation nodes are Tier 1 links (*cf.*, Fig. 5.1) and have a capacity of 2 Gbps and transmission delay of 1.8 ms. Other links are called Tier 2 links and have a capacity of 3 Gbps and a delay of 4.8 ms [8].

5.1.1 Implementation

We used RLlib's [59] implementation of PPO, alongside Ray's Tune platform [60] for training and hyper-parameter optimization. The training was carried out on an NVIDIA A100 GPU and took 139 hours. For hyper-parameter optimization, we utilized the population-based training (PBT) technique [61] and final results were achieved using the values in Table 5.5. Inspired by genetic algorithms, in PBT, multiple neural networks are trained in parallel using varying hyperparameters. After a predefined period of training, the best-performing model's weights are copied across the rest of the models and the hyperparameters are refined to employ both exploitation and exploration.

Table 5.5: GNN-AC-SL training hyper-parameters

Parameter	**AC**	**Slicing**
PPO clipping	0.05	0.2
entropy coefficient	0.1	0.0001
gamma	1	0.99
lambda	1	
learning rate	1e-05	
gradient clipping	10	
SGD iters	50	
batch size	372000	

5.2 Baselines

The following heuristic and DRL-based methods are implemented for comparison. DRL-AC and MLP-AC-SL are the only baselines with an intelligent AC module. Others greedily admit all feasible SRs.

- **Centralized:** a heuristic that places VNFs as close to the core node as allowed by delay and capacity constraints. In this approach, first, a path is drawn from the access node where the SR arrives to the closest core node. The physical nodes in this path are sorted from the core to the access node, and the VNFs in the chain are sorted from the last (MEC) to the first (RU). At each step, the current VNF in the list is placed and the algorithm moves on to the next VNF, if the current physical node has enough capacity to host it. Otherwise, the algorithm moves on to the next node in the list. If, all the VNFs are placed before going through all physical nodes, the slice request is considered successfully deployed, else, it is rejected and the algorithm moves on to the arrival of the next slice request.

- **Node-Ranking (NR):** a heuristic for isolation-aware RAN slicing with delay constraint. We consider the scenario of the highest isolation level in [37], with the objective of minimizing active sites. In this approach, all the physical nodes are ranked according to their CPU, RAM, and bandwidth usage, as well as their delays from the originating access node and their tiers. The highest ranking node is selected to host the current VNF. If there is not enough capacity on the selected nodes to host their respective VNFs, the slice request is considered rejected.

- **GRC:** a heuristic approach that proposes a novel metric, called global resource capacity (GRC), to rank the embedding potential of each substrate node [62]. After ranking the physical nodes, the algorithm applies greedy load-balancing to embed each virtual node sequentially on the first physical node with enough available processing capacity. Next, the shortest path routing is adopted to create a path for embedding each virtual link. In order to optimize this work for metro 5G slicing, we add the RAM capacities into consideration by averaging CPU and RAM, when dealing with node capacities. Moreover, we only select among the physical nodes that satisfy the delay and bandwidth constraints.

- **DRL-AC:** slicing is performed using the Centralized algorithm, while the admission is decided by the DRL-based AC agent.

- **[MLP/GNN]-SL:** in these two approaches, slicing is managed by a DRL agent based on MLP/GNN architecture. However, all feasible slices are admitted and deployed, *i.e.*, there is no admission policy.

- **MLP-AC-SL:** multi-agent DRL-based method that uses MLP architecture for both agents similar to [1].

Additionally, to calculate the upper-bound in terms of the achieved revenue of the joint slicing and AC solution in the training scenario, we utilize the Gurobi Optimizer for solving the MILP defined in Section 3.3.

5.3 Results

5.3.1 Training

We train each model for 180 million training steps and plot the progress in terms of the total revenue achieved per episode in Fig. 5.2. The MILP solution achieves a revenue of 99563 with an accuracy of 93.85%. However, the MILP solution must work in an offline manner, *i.e.*, when all the SRs are known in advance. This is not applicable to a real-life scenario where the SRs arrive over time and without previous knowledge. This removes the required prediction from the problem, which simplifies it by a great margin, resulting

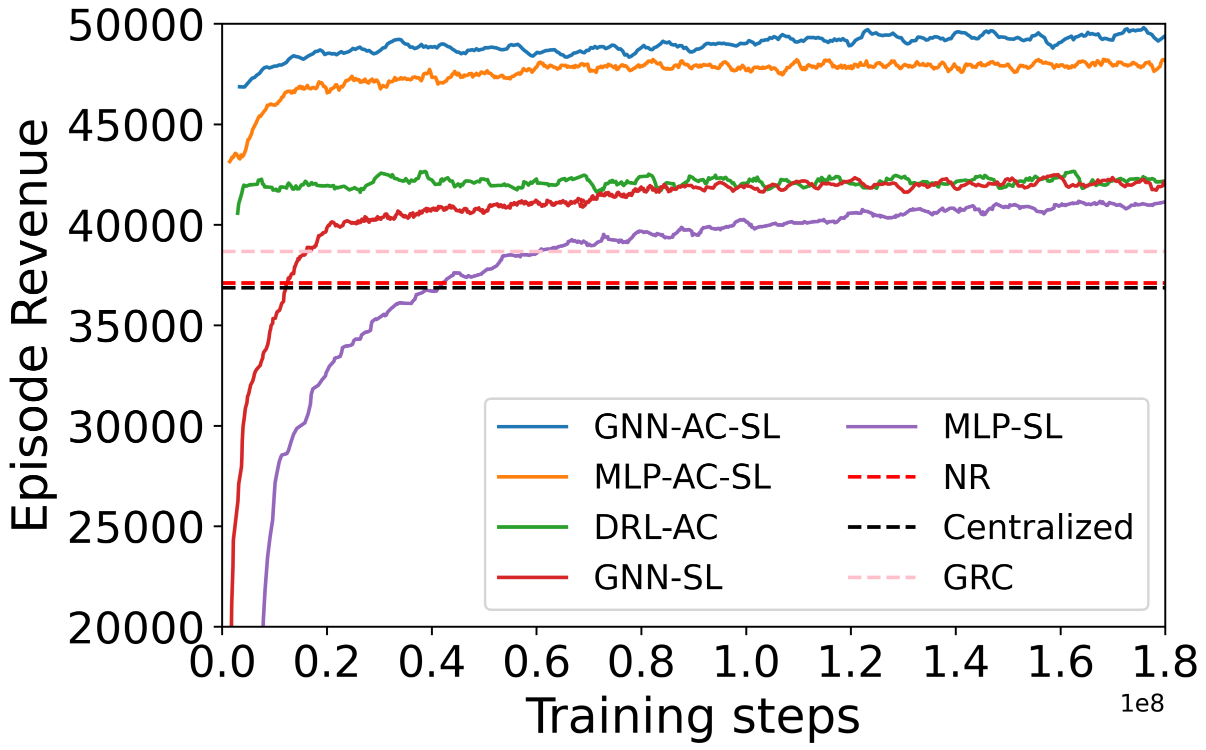


Figure 5.2: Episode revenue during training

in the huge gain in revenue compared to the DRL-based approaches. Moreover, it is computationally infeasible as each evaluation takes 10 hours on an 8-core 3.3 GHz Intel Xeon CPU. Even though the training of the DRL approaches takes 139 hours, in the end we have models that are able to generate slicing and AC decisions for any scenario in the order of sub-seconds. In contrast, the MILP takes 10 hours to generate slicing and AC decisions for a single set of 2000 SRs. Therefore, the MILP result is presented only as a theoretical upper-bound of the achievable revenue in this evaluation scenario and should not be compared with the rest of the approaches as a practical solution.

Since GNNs are optimized for graph data, GNN-based approaches, *i.e.*, GNN-AC-SL and GNN-SL, have faster convergence and higher performance compared to their MLP-based counterparts, respectively. Moreover, DRL-AC converges the fastest due to its small action space, albeit to a local optimum, as the slicing performance of the Centralized approach prevents it from reaching its full potential. However, once the intelligent AC and slicing agents are combined, they can operate in harmony and reach their maximum ca-

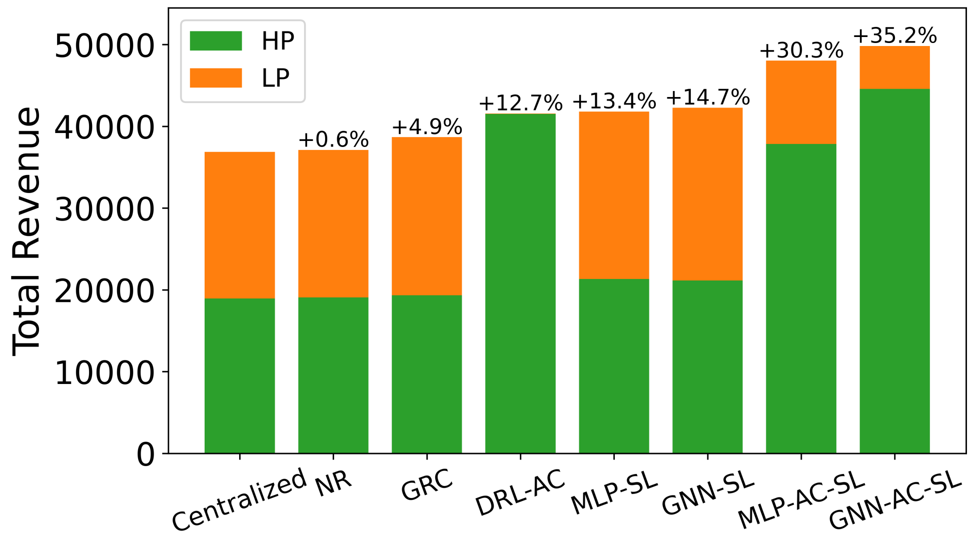


Figure 5.3: InP revenue with improvement relative to Centralized. Each bar also displays the portion of LP/HP SRs admitted.

pacity. Moreover, all the DRL-based approaches outperform the heuristic approaches once sufficiently trained.

5.3.2 Evaluation

Once the training ends, the best performing model checkpoint is restored to perform the evaluations. For evaluation, the exploration is disabled, changing the action selection policy from stochastic sampling to deterministic (*i.e.*, selecting the action with the highest probability), thereby improving the revenue margins slightly. In Fig. 5.3, the total revenue and the portion of HP and LP SRs admitted are shown. The percentage improvement of all approaches relative to Centralized is also shown above each bar.

Among the baselines, the Centralized approach performs the worst, as it greedily places near the core, leading to bandwidth bottlenecks, especially with high-throughput SRs and in the more crowded links. Moreover, this approach tends to ignore potential hosts nearby, even if they have the capacity, as it only considers the nodes in the shortest path towards the core. Next, NR performs relatively similarly since ranking works on a feature weighting basis that requires manual fine-tuning based on SR type and substrate network, rendering it inefficient for different networks. In this regard, GRC performs better, since its ranking

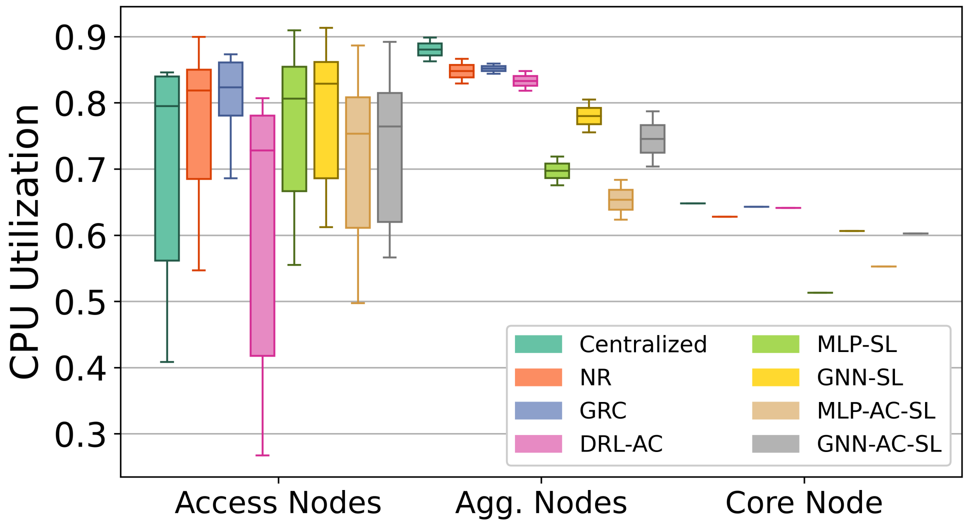


Figure 5.4: CPU utilization across the network nodes. It should be noted that there is a single core.

approach does not require manual adjustment and their global resource capacity metric is generalizable.

The heuristic approaches are followed by DRL-based methods. Although DRL-AC takes advantage of an intelligent AC, it is stuck on a local optimum, deploying high-priority SRs only. This is a result of the inability of the Centralized approach to avoid bottlenecks and accommodate more SRs. Consequently, with inefficient embeddings and less overall SR embeddings offered to the AC agent, it chooses to skip LP SRs completely. Next, we have the two slicing-only approaches that follow the same pattern as observed in Fig. 5.2, with GNN-SL achieving a slightly higher revenue than MLP-SL. These approaches deploy the same number of HP SRs as LP SRs, since they greedily admit all feasible SRs. Following, are the two joint slicing and AC solutions. As GNN-SL is shown to be a more efficient slicing solution compared to MLP-SL, the AC agent in GNN-AC-SL is also able to admit more HP SRs in the network without creating bottlenecks. Such advantage makes GNN-AC-SL the highest achieving approach in terms of the overall gained revenue, with 35.2% higher revenue when compared to Centralized.

Figures 5.4 and 5.5 compare the distribution of links and nodes utilization across the substrate network. The Centralized approach has the highest utilization of higher-tier nodes, however, it falls behind DRL-based slicing algorithms when considering access

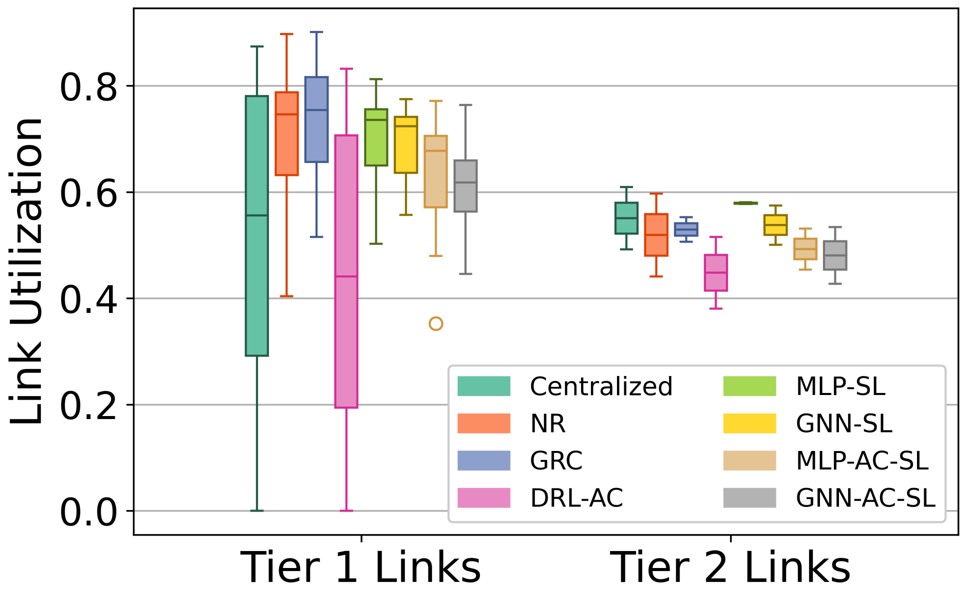
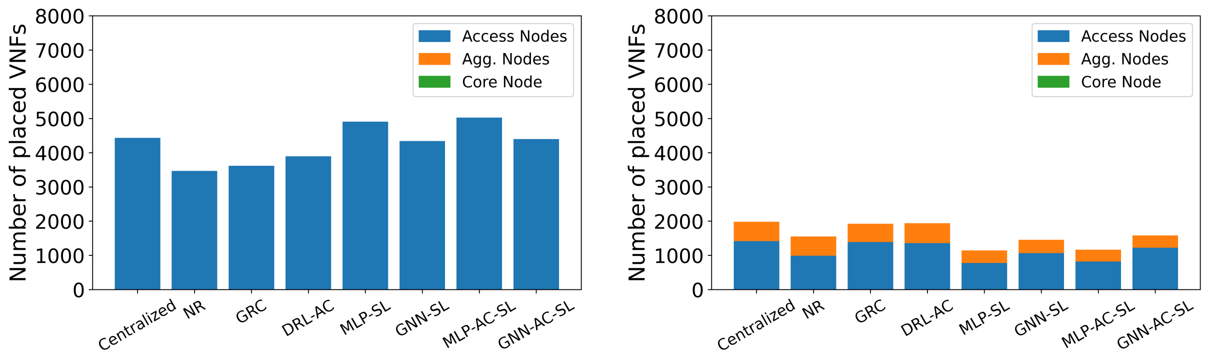


Figure 5.5: Link bandwidth utilization across the network. Links connecting the Core node to the Main nodes are Tier 2, while the rest are Tier 1.

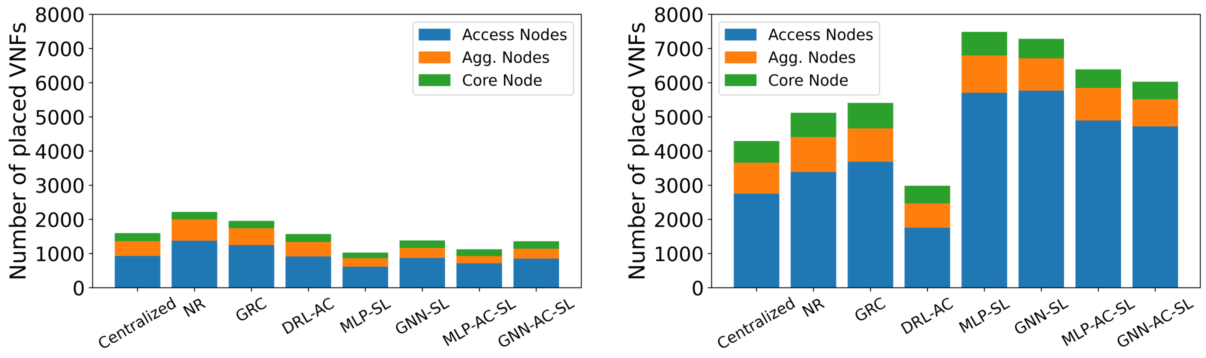
nodes. In fact, Centralized and consequently DRL-AC have the highest variation of utilization across access nodes. This shows that the Centralized method is not able to balance the load across the substrate network and utilize the available capacity in the access nodes. In contrast, GRC displays less variance in its node/link utilization, which is a result of its superior node ranking approach leading to a better load-balancing performance. Furthermore, GNN-based solutions exhibit a higher utilization of higher-tier nodes when compared to the MLP-based methods without creating link bottlenecks. Finally, the addition of the AC module generally lowers the utilization to reserve space for prospective HP SRs.

Fig. 5.6 shows an in-depth look at slicing decisions of each method for all SR types, *i.e.*, MEC applications. VS and RS SRs are deployed more than VR and CG in all approaches, as they are less demanding in terms of processing capacity. RS SRs have a restrictive delay tolerance which limits their placement to their originating access nodes. On the other hand, VS SRs have the most utilization of the aggregate and core nodes due to their relaxed delay requirements. Compared to MLP-based approaches, GNN-based methods aim to increase the deployment of higher-paying CG and VR SRs at the cost of less SRs of type RS and VS. In this regard, DRL-AC rejects too many VS SRs when compared to MLP-AC-SL and GNN-AC-SL.



(a) Remote Surgery (RS)

(b) Cloud Gaming (CG)



(c) Virtual Reality (VR)

(d) Video Streaming (VS)

Figure 5.6: VNF placement of different methods for each slice type, *i.e.*, MEC application

5.3.3 Robustness and Generalizability Analysis

5G networks observe different trends over time, therefore, it is imperative that the utilized orchestration algorithms are able to cope with changing network conditions. In this section, we evaluate the robustness and generalizability of the trained agents by deviating the evaluation conditions from the training scenario. As previously mentioned, during training, the proportion of HP slice requests arriving is set to 50%. Fig 5.7 plots the impact of changing this proportion on the total InP revenue for different approaches.

Since slicing-only approaches are imperceptive to the revenues offered by the slice requests and are tasked only with proposing successful slice embeddings, their total revenue stays linearly relative to the proportion of the HP requests. In addition, as the proportion

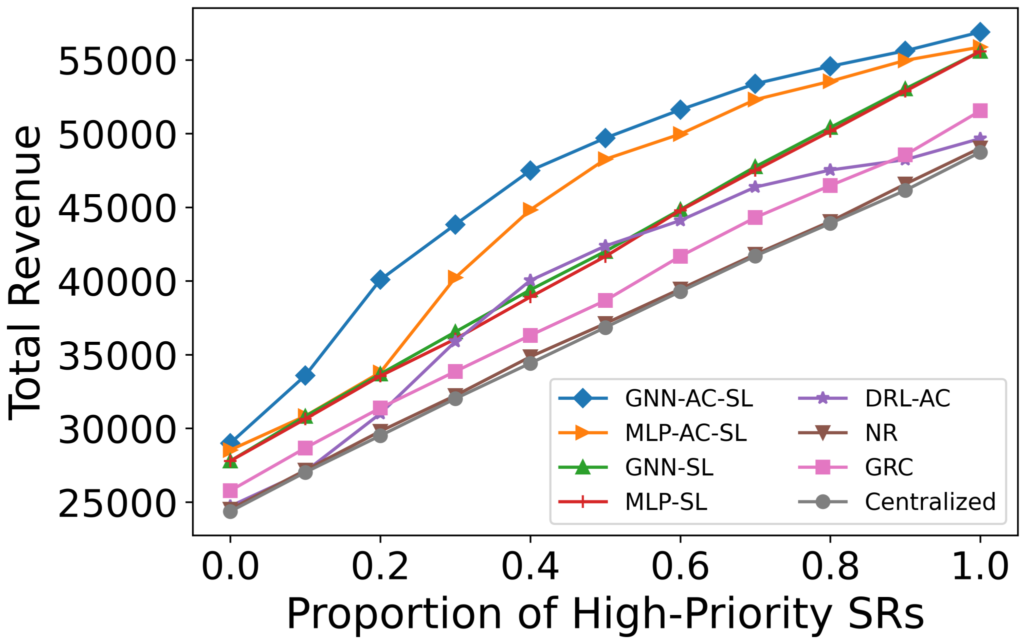


Figure 5.7: Revenue vs. HP SR proportion

of HP SRs approaches 0 or 1, the AC effect diminishes, resulting in GNN-AC-SL, MLP-AC-SL, and DRL-AC performing similarly to their slicing-only counterparts. Furthermore, with few HP SRs in the lower end of the spectrum, the difference between MLP-AC-SL and GNN-AC-SL is more prominent, which is due to a better slicing performance, as previously demonstrated.

Fig. 5.8 compares algorithms under different network loads resulting from changing SR arrival rate. We observe that GNN-AC-SL maintains its superior performance consistently. Moreover, as the arrival rate decreases, MLP-based approaches degrade significantly. This corroborates the fact that MLP models can over-fit to the input and are less generalizable in graph applications, when compared to GNNs, to the extent that with an arrival rate of 0.2 SRs/timestep GNN-AC-SL outperforms MLP-AC-SL by 24%. Additionally, with an arrival rate of 0.2 SRs/timestep, the scenario becomes trivial, as evident by the similar performance of NR and GRC methods to DRL-based methods DRL-AC and GNN-SL.

Fig. 5.9 plots the achieved revenue of the algorithms under different network link capacities as a result of multiplying the bandwidths of all physical links by a coefficient. As it can be observed the approaches follow the same performance patterns as Fig. 5.8, where

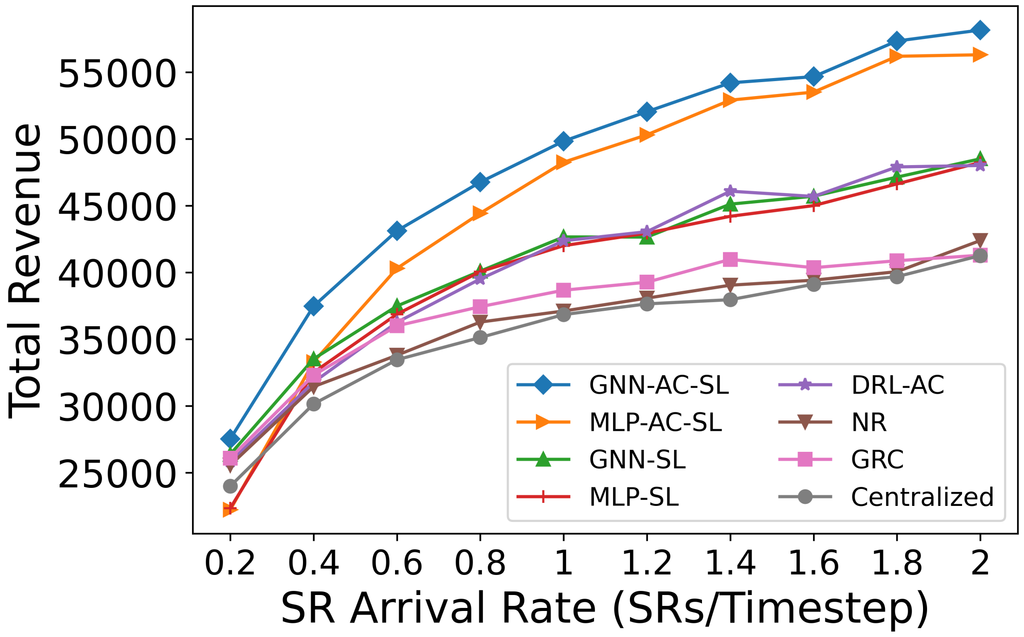


Figure 5.8: Revenue vs. SR arrival rate

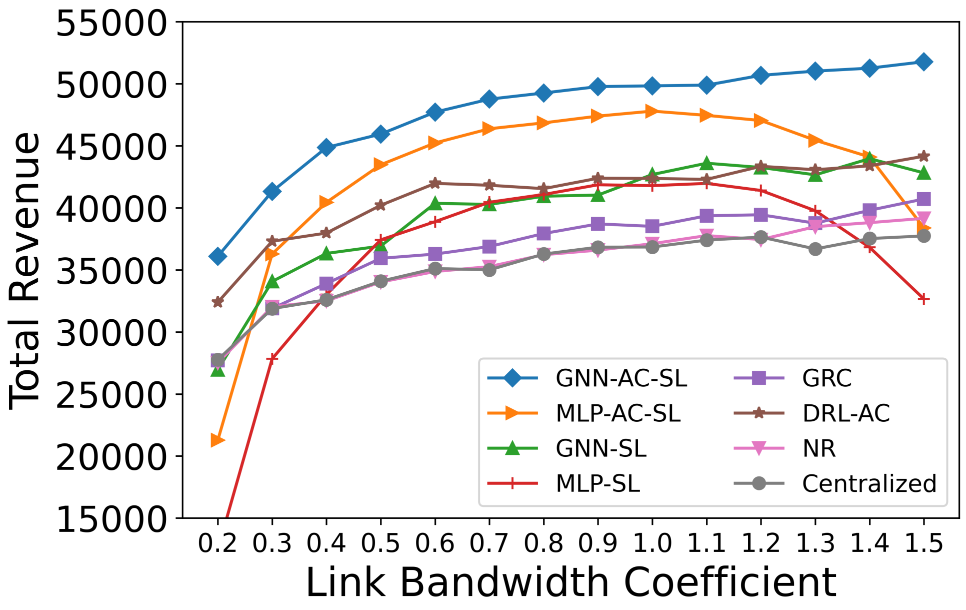


Figure 5.9: Revenue vs. changing link bandwidths

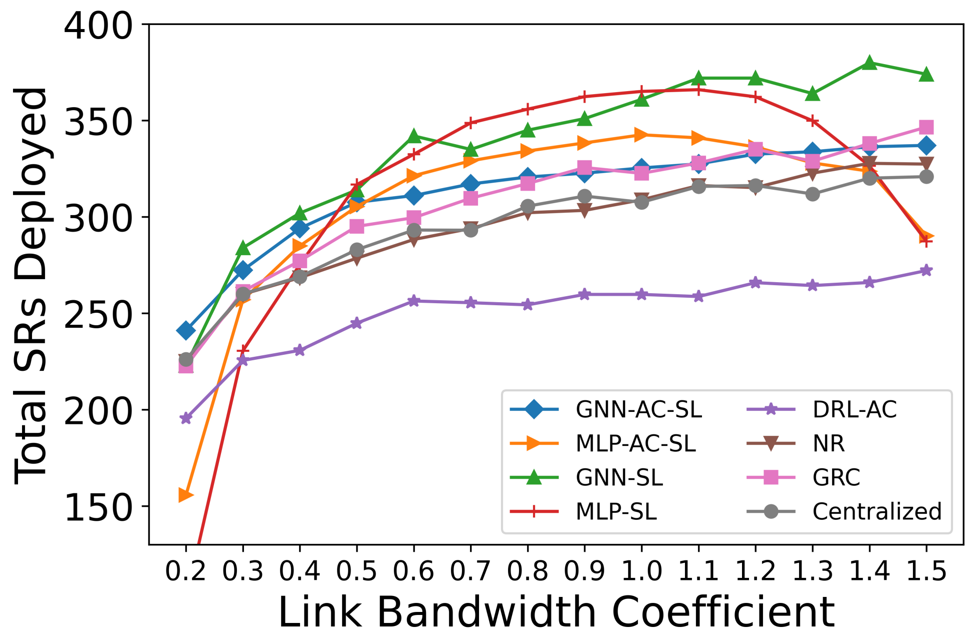


Figure 5.10: Total deployed SRs vs. changing link bandwidths

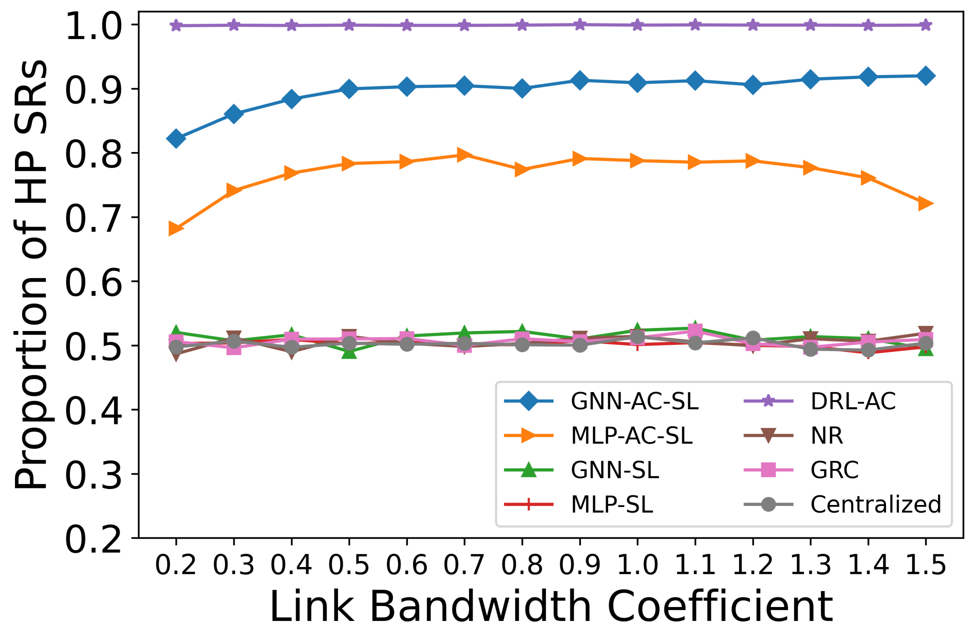


Figure 5.11: Proportion of HP SRs deployed vs. changing link bandwidths

GNN-AC-SL holds the highest achieved revenue consistently, while the MLP-based slicing models MLP-SL and MLP-AC-SL lose significant performance when deviating from their training scenarios. GNN-AC-SL outperforms MLP-AC-SL by 34.9% when the link capacities are set to 1.5$\times$ of the values used for training the model. In MLP-based models, the input is fixed in size and order, therefore, each input neuron overfits to the training data and its range. Since the link bandwidths are set to 2000 Mbps and 3000 Mbps for Tier-1 and Tier-2 links, respectively, the DNN is trained on a relatively small range of values (*i.e.*, 0-2000 and 0-3000 for neurons connected to Tier-1 and Tier-2 links' input data, respectively). Multiplying these values by 1.5 results in numbers far from what the model was trained on, propagating errors throughout the model and resulting in a collapsed network output. This problem is less noticeable in GNN-based architectures, where the same input neurons are fed with every link and node's data. This creates a more generalizable model that is able to deal with a greater range of input values.

Moreover, with less link capacities, there is a higher chance of link bottlenecks, which could potentially be avoided by smart admission control. Therefore, the DRL-AC approach does not lose as much revenue in lower link bandwidths when compared to the slicing-only models. We plot the total deployed SRs and the proportion of HP SRs in Figures 5.10 and 5.11, respectively. As it can be observed, the drops in achieved revenue of the MLP-based models are due to a lower number of deployed SRs resulting from poor slicing decisions.

Following Fig. 5.9, in Fig. 5.12, we analyze the robustness of the algorithm against varying node capacities, by multiplying the resource capacities (*i.e.*, CPU and RAM) of all the substrate nodes by a coefficient. In these experiments, GNN-based approaches maintained their performance levels even in network deviations, while MLP-SL and MLP-AC-SL methods faced difficulties in lower node capacities. In the case of 1/2 overall node capacity, GNN-AC-SL outperforms MLP-SL and MLP-AC-SL by 35.3% and 16.8%, respectively. The number of total deployed SRs and the proportion of deployed HP SRs are shown in Figures 5.13 and 5.14, respectively. It should be noted that even though GNN-SL and MLP-SL have the highest number of deployed SRs, their total achieved revenues remain lower than the joint slicing and admission control solutions which utilize a higher proportion of HP SRs.

To showcase the generalizability of our proposed model on different topologies, we evaluate the models on unseen substrate networks, without re-training or re-purposing the model. We only consider the heuristic and GNN-based approaches on these networks, since MLP-based models are tied to a fixed input size and the previously trained MLP-based methods cannot operate if the network changes. We consider a moderate-sized 31-node synthetic network, and the 52-node real Milan network [6] shown in Fig. 5.16, which are 2.5$\times$ and 4$\times$ the size of the training network, respectively and have resource capacities and

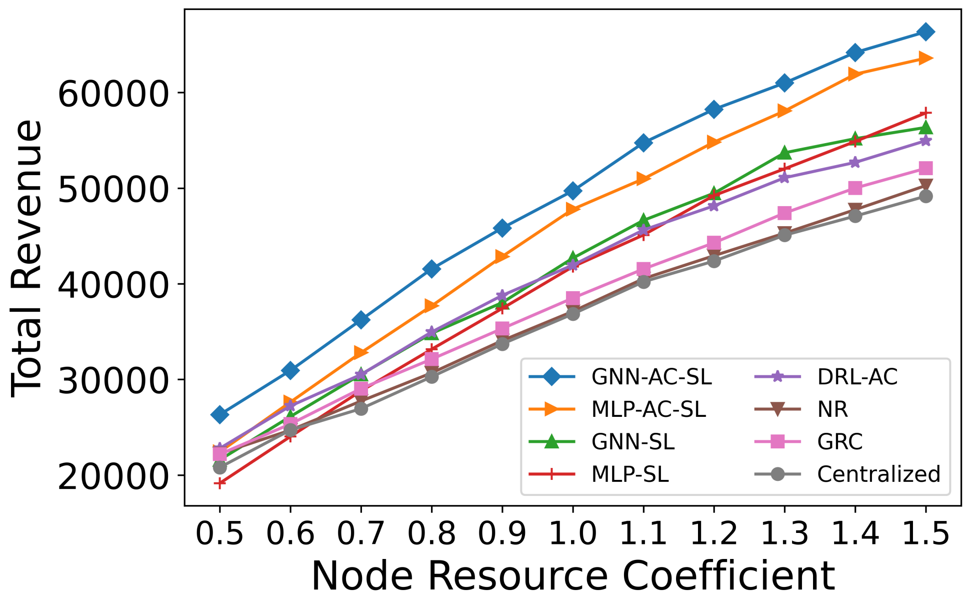


Figure 5.12: Revenue vs. changing node CPU and RAM capacities

delays similar to it. Fig. 5.15 plots the performance of the heuristic and GNN-based approaches on these networks, and as evident, GNN-AC-SL outperforms the heuristic baseline by 25.5% and 25.1% in the synthetic and Milan topologies, respectively. However, with the larger size and overall capacity of these networks, there are less bottlenecks. Consequently, the impact of the AC approach is less notable and the percentages of improvements compared to the Centralized baseline are lower. In general, these results confirm that while trained on a small-scale network, our proposed GNN-AC-SL solution scales well to larger real-world metropolitan networks.

Finally, we examine the robustness of the algorithms against dynamic network topologies, *e.g.*, when nodes and links are disabled due to network shutdown. We Simulate scenarios where nodes and their connecting links are disconnected from the network in five stages. At each stage, the respective highlighted section in Fig. 5.17 is added to the previously failed nodes, increasing the magnitude of the failure. The impact of these changes is shown in Fig. 5.18. Since the heuristic approaches are not biased towards a specific topology, they maintain a smooth loss in revenue. However, GNN-AC-SL and GNN-SL models exhibit irregularities in their achieved revenue. This could be contributed to specific bottlenecks that happen due to a shift in slice arrival location distribution. Even though

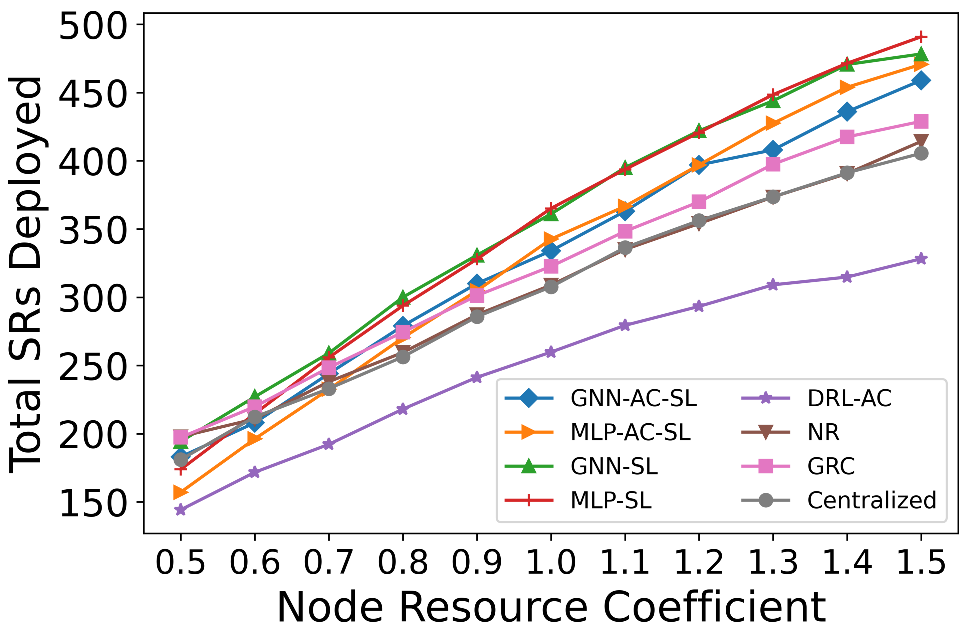


Figure 5.13: Total deployed SRs vs. changing node CPU and RAM capacities

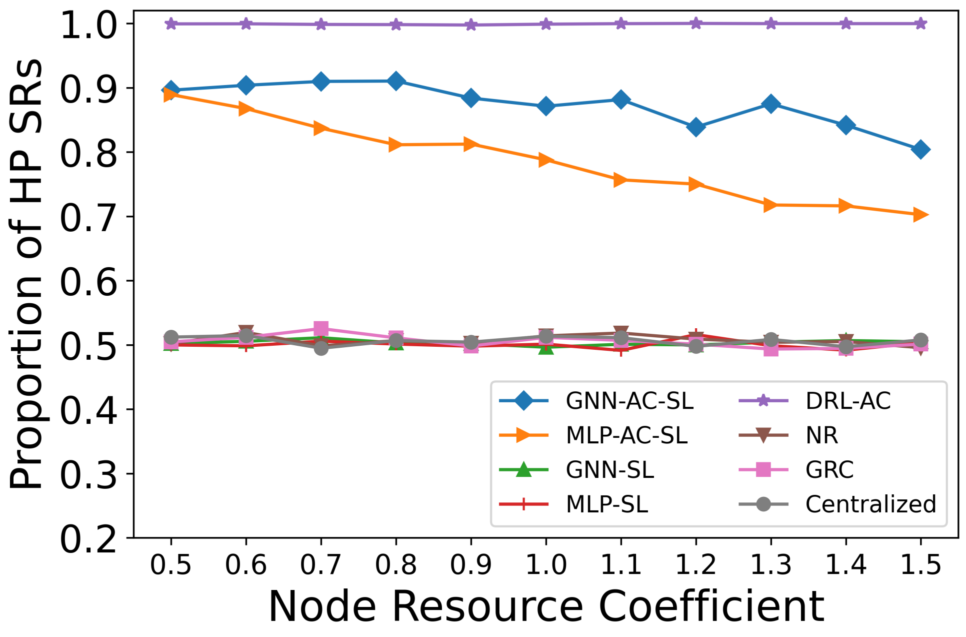


Figure 5.14: Proportion of HP SRs deployed vs. changing ndoe CPU and RAM capacities

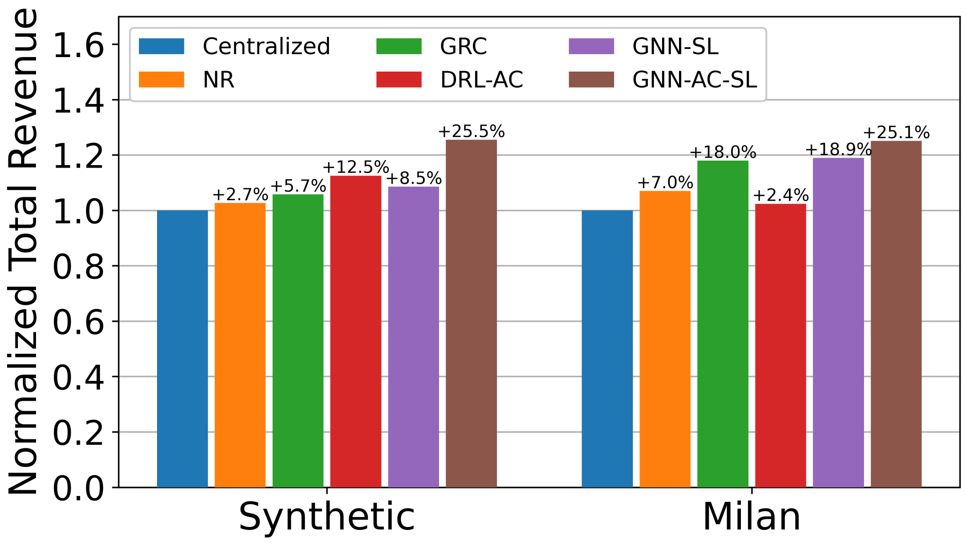


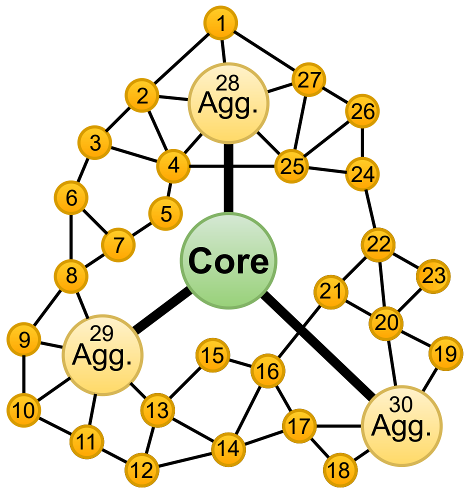
Figure 5.15: InP revenue in large previously unseen networks

GNN-AC-SL maintains the highest performance, a more generalizable model would be required to create more predictable outcomes in such cases.

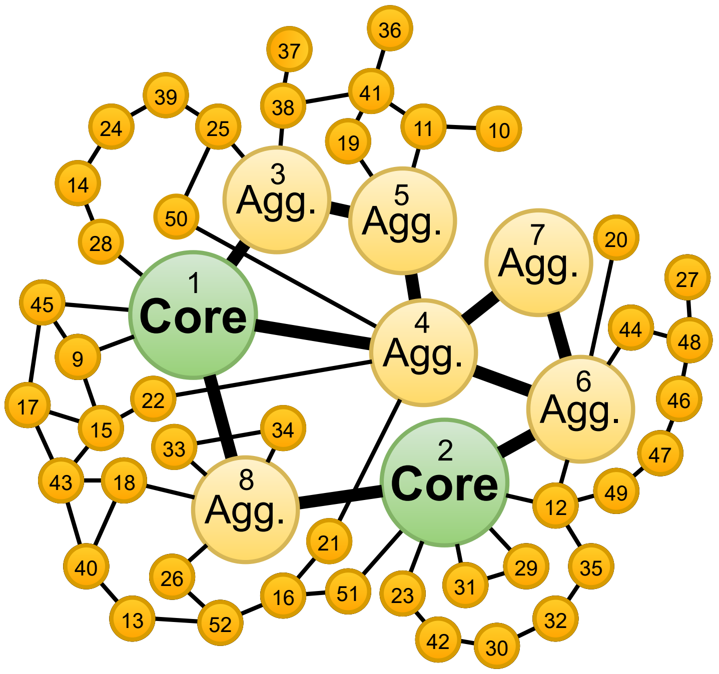
5.3.4 Attention Map Analysis

As previously explained, GATs are based on the attention mechanism, which can be utilized to extract important sections of the graph. In 5G infrastructure, the InP can make use of this opportunity to increase the capacity in critical areas identified by analyzing the attention scores from all neighboring nodes. Fig. 5.19 is generated from the attention layer of the slicing agent when operating on the training topology, by calculating the average of the attention scores of the last GATv2 layers towards each node at each step. The node indices follow Fig. 5.1. As evident, nodes 1, 2, and 3 pay ample attention to nodes 6, 5, and 4, respectively, as they are leaf nodes and their states depend a lot on their only neighbors. Furthermore, nodes 5 and 6 receive the highest attention scores among non-core nodes, as they are critical nodes that bridge the two halves of the network. A bottleneck in any of these nodes can prevent slices from crossing over to more desirable nodes on the other side of the network. Finally, the core node has the second highest attention scores, as it plays a critical role in hosting computation-heavy applications due to its larger capacity.

Similarly, we extract the attention maps of the Synthetic and Milan topologies, which



(a) Synthetic Topology



(b) Milan Topology

Figure 5.16: New studied topologies: (a) is the synthetic network with 31 nodes and 54 links, and (b) is the Telecom Italia metro-regional network [6] with 52 nodes and 72 links.

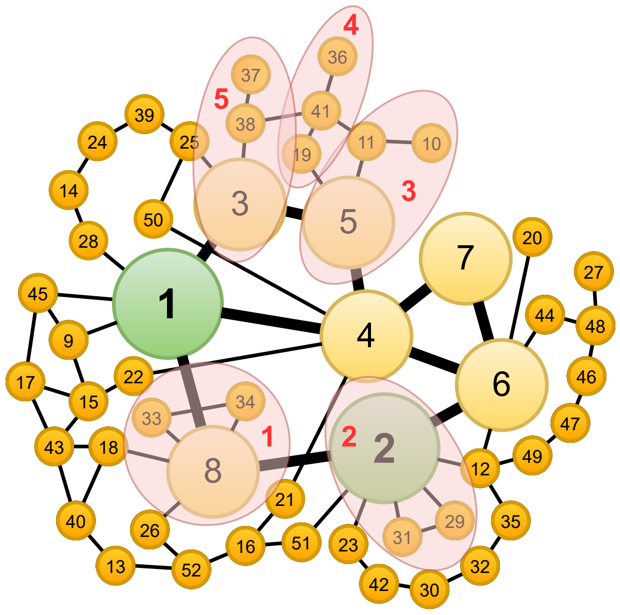


Figure 5.17: Milan topology failure scenario. Failing links and node are highlighted in red.

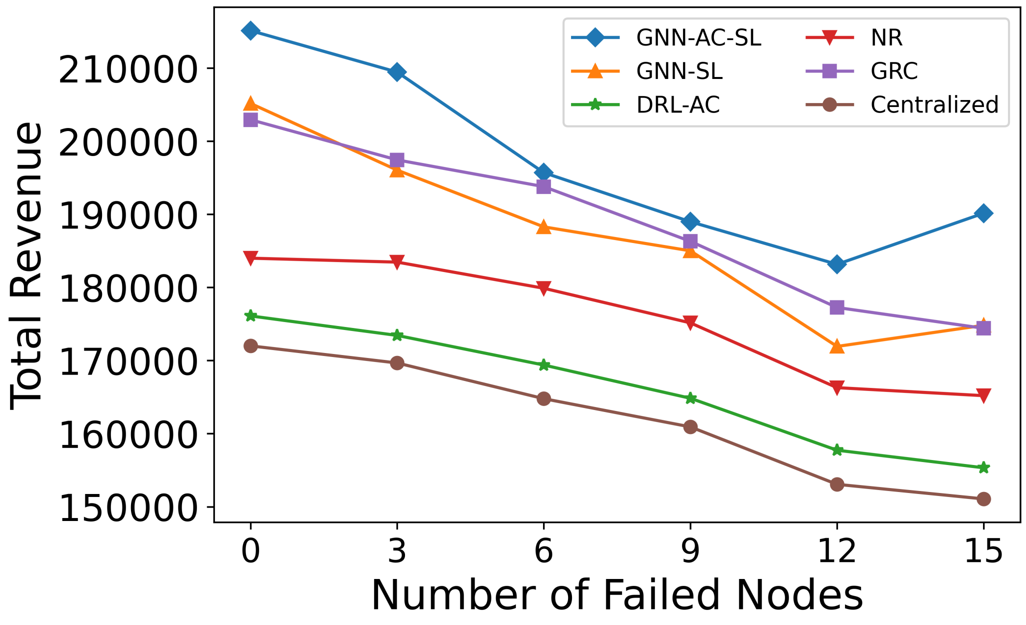


Figure 5.18: Impact of network failure on achieved revenue

can be seen in Figs. 5.20 and 5.21, respectively. The goal of the final GATv2 layer in our slicing agent's model is to select the most suitable substrate node for hosting the current VNF. Therefore, it will attend to nodes with more ideal states (*e.g.*, higher available capacities). We can utilize this fact to improve the capacities at less attended nodes, indicating that they are currently facing bottlenecks. We assess the significance of these attention values by changing the node capacities of the 10 most and least attended nodes (excluding the core nodes) in the Milan topology and observing the changes in the achieved revenue. We multiply these nodes' resource capacities (*i.e.*, CPU and RAM) and their connecting links' bandwidths by a coefficient and display the results in Figures 5.22 and 5.23. As expected, by focusing the resources at the less attended nodes, the InP experiences a higher gain in revenue when compared to the highest attended nodes. Moving from a resource coefficient of 0.5 to 1.5 results in a 8.8% and 9.9% improvement in the total achieved revenue for the 10 most attended nodes with the Centralized and GNN-AC-SL approaches, respectively. In contrast, these values are 23.8% and 14.4% for the 10 least attended nodes, denoting a higher gain in revenue for a similar cost of upgrading the resources.

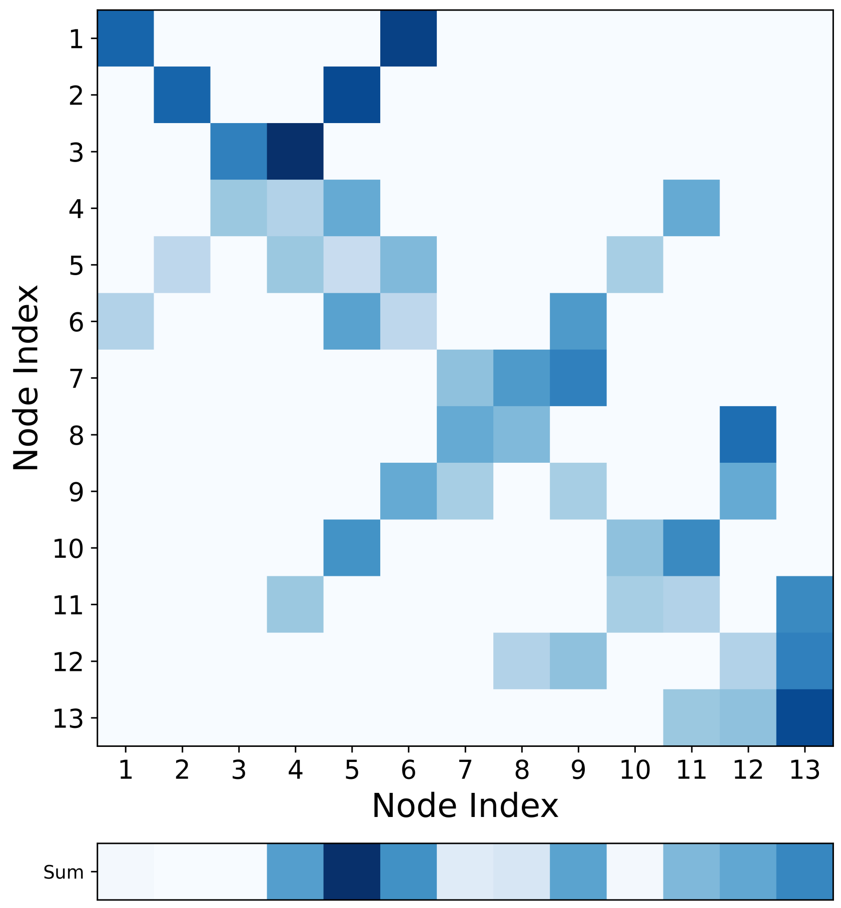


Figure 5.19: Extracted attention map of the slicing agent's GATv2 layer when operating on the training topology (Fig. 5.1). Some values are set to 0, since GATv2 calculates the attention scores for pairs of neighboring nodes only.

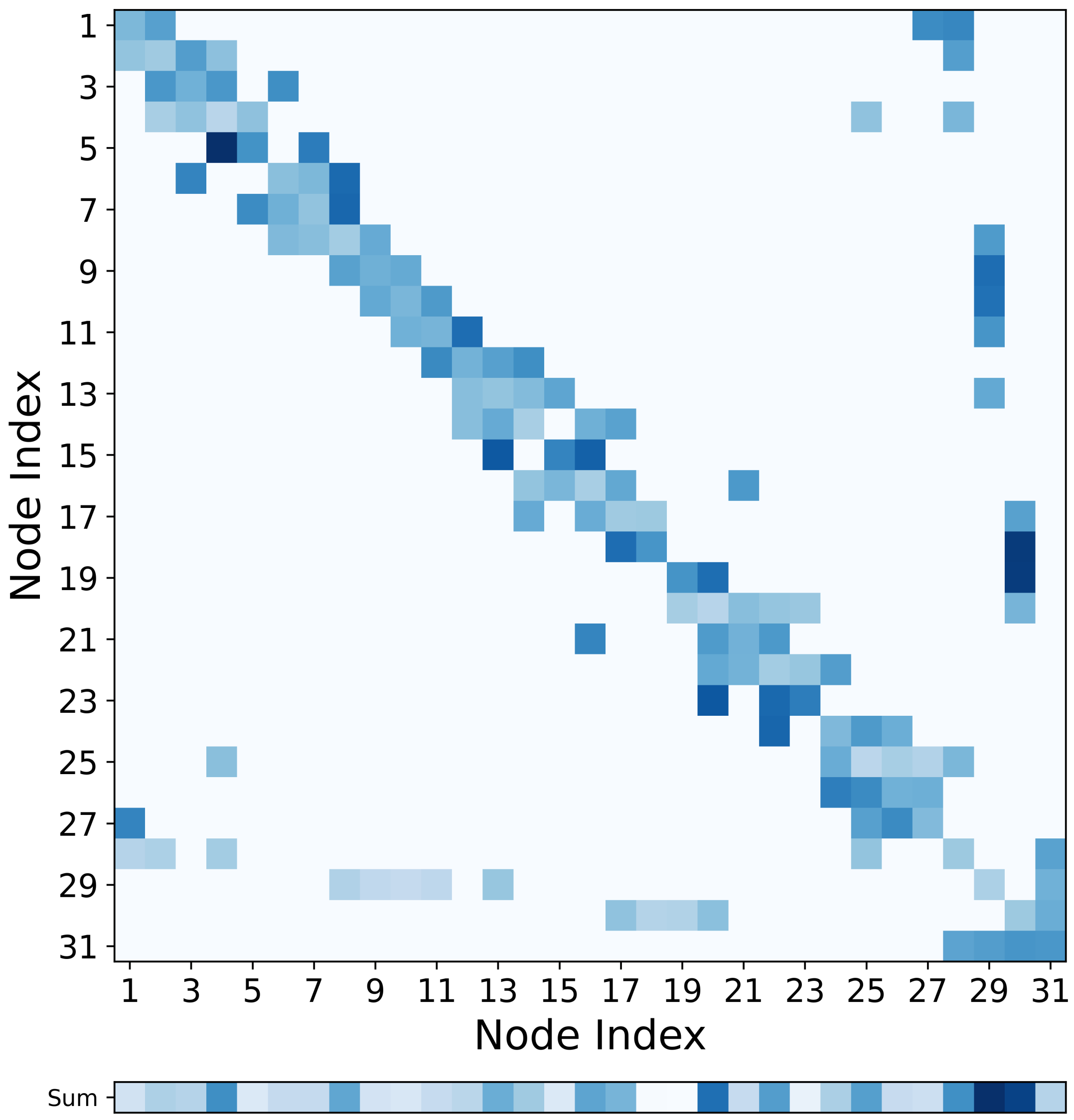


Figure 5.20: Extracted attention map of the slicing agent's GATv2 layer when operating on the Synthetic topology (Fig. 5.16a)

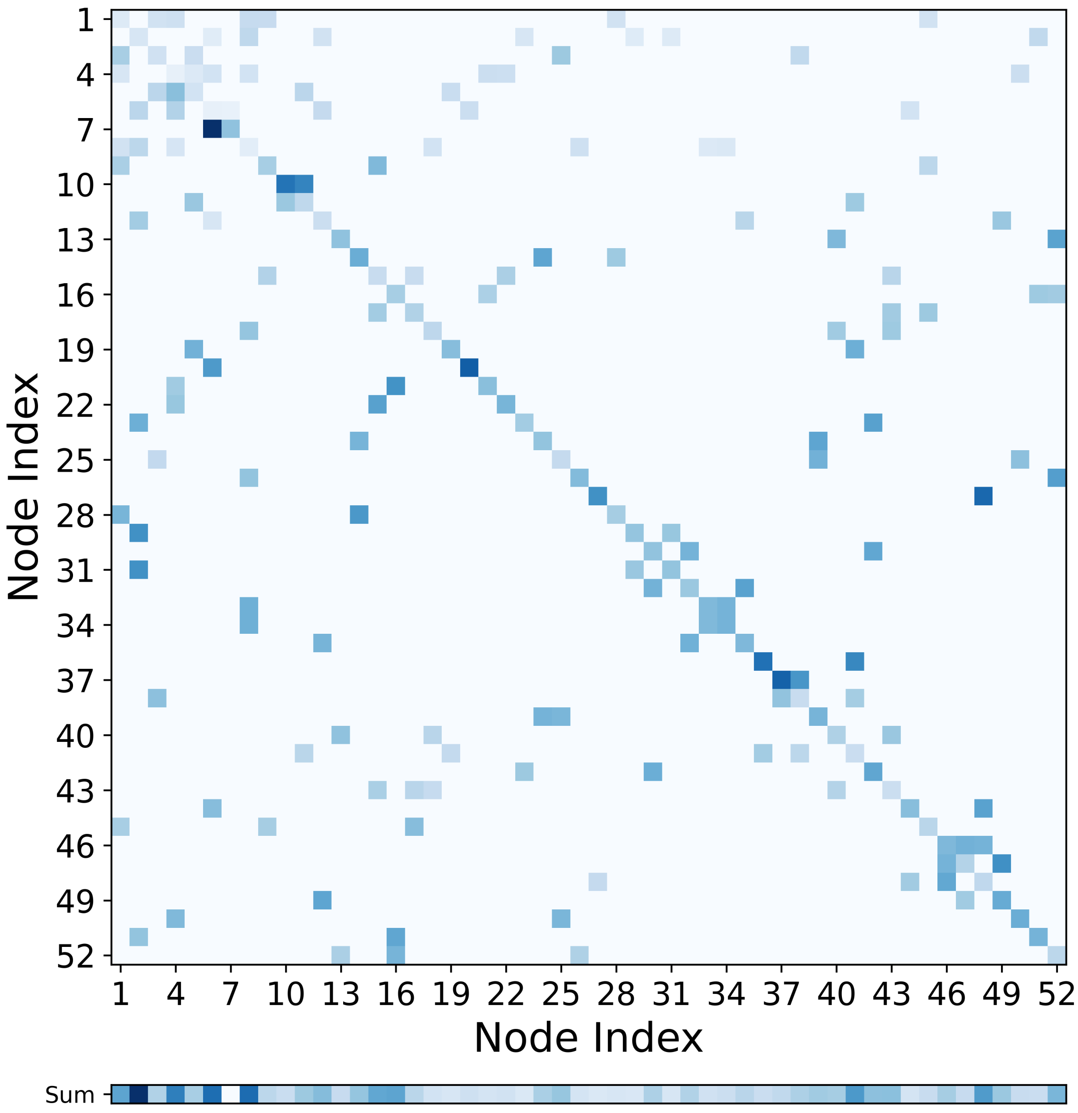


Figure 5.21: Extracted attention map of the slicing agent's GATv2 layer when operating on the Milan topology (Fig. 5.16b)

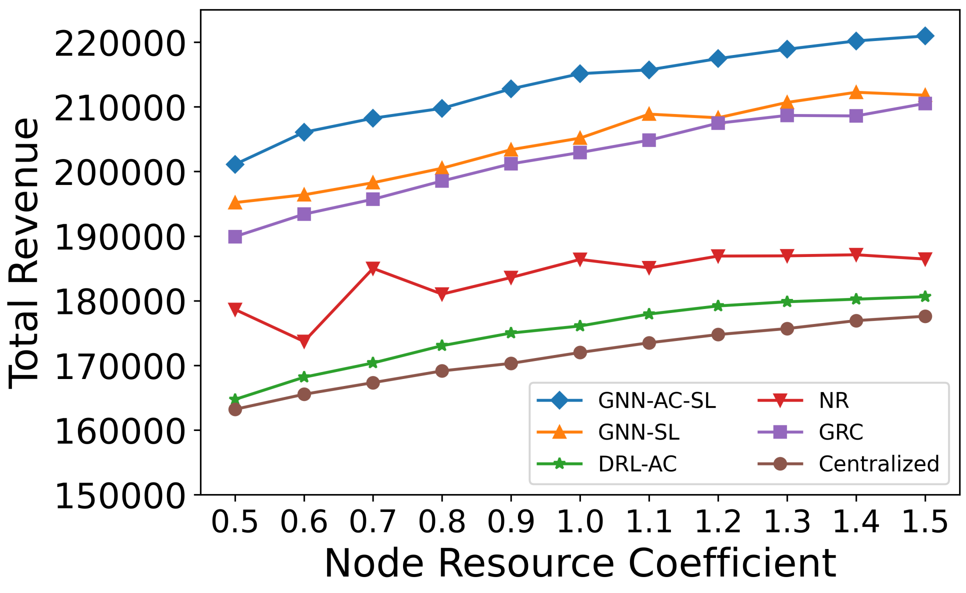


Figure 5.22: Revenue vs. changing capacities of the 10 most attended nodes and their connecting links in the Milan topology (Fig. 5.16b)

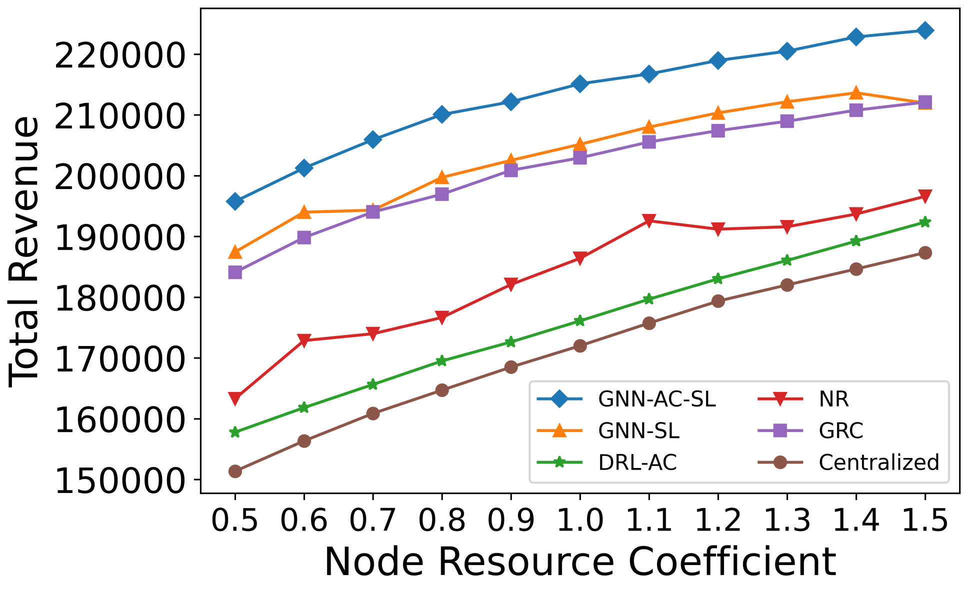


Figure 5.23: Revenue vs. changing capacities of the 10 least attended nodes and their connecting links in the Milan topology (Fig. 5.16b)

Chapter 6

Conclusions and Future Work

6.1 Conclusion

In this work, we addressed the problem of joint RAN/MEC slicing and AC in dynamic metropolitan 5G networks using multi-agent DRL. Conventional DRL approaches in this area are based on fully connected NN designs that are bound to the input/output data dimensions and order, and as such, they fail to operate on topologies different than what they were trained for. We proposed a GNN-based solution that can effectively scale to large previously-unseen networks without the need for re-training. The results showed that our proposed solution converges faster than the MLP-based counterparts and achieves as much as 35.2% and 25.5% higher overall revenue, when compared to heuristic baselines, on the training and unseen networks, respectively. Moreover, we showed that DRL-based approaches that address only one aspect of the problem lead to a loss in potential InP revenue. Additionally, our robustness and generalizability analysis showed that GNN-based solutions for RAN slicing are more robust, due to the graph-based nature of metropolitan 5G infrastructures and GNNs' ability to learn the complex structures of graphs. Finally, we showed that using GATv2 allows for the extraction of an attention map that can assist InPs in identifying network bottlenecks and efficiently improving network performance.

6.2 Future Work

As robust as the current solution is, it could still potentially benefit from varying substrate networks and node/link capacities during training. A possible direction for future work is investigating the impact of training on random topologies and scenarios on generalization. Furthermore, we can investigate dynamically scaling the resources allocated to SRs based on their predicted demand. This can allow the network to accommodate more SRs by leveraging over-booking.

References

[1] M. Sulaiman, A. Moayyedi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Multi-agent deep reinforcement learning for slicing and admission control in 5G C-RAN," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[2] M. Sulaiman, A. Moayyedi, M. Ahmadi, M. A. Salahuddin, R. Boutaba, and A. Saleh, "Coordinated slicing and admission control using multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.

[3] "Study on new radio access technology: Radio access architecture and interfaces," *3GPP TR 38.801 V14.0.0 (R14)*, 2017.

[4] "5G wireless fronthaul requirements in a passive optical network context," *ITU-T G Suppl. 66*, 2020.

[5] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "MEC in 5G networks," *ETSI white paper*, vol. 28, 2018.

[6] L. Askari, F. Musumeci, and M. Tornatore, "Latency-aware traffic grooming for dynamic service chaining in metro networks," in *IEEE ICC*, 2019, pp. 1–6.

[7] "Service requirements for the 5g system; stage 1," *3GPP TS 22.261 V18.6.0 (R18)*, 2022.

[8] "NGMN overview on 5G RAN functional decomposition," *NGMN Alliance, Final Deliverable, Feb. 2018, version 1.0.*, 2018.

[9] "Characteristics of transport networks to support IMT-2020/5G," *ITU-T G.8300*, 2020.

[10] M. Shehata, A. Elbanna, F. Musumeci, and M. Tornatore, "Multiplexing gain and processing savings of 5G radio-access-network functional splits," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 4, pp. 982–991, 2018.

[11] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vRAN/MEC orchestration," in *IEEE Conference on Computer Communications*, 2018, pp. 2366–2374.

[12] A. Garcia-Saavedra, G. Iosifidis, X. Costa-Perez, and D. J. Leith, "Joint optimization of edge computing architectures and radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2433–2443, 2018.

[13] F. W. Murti, S. Ali, and M. Latva-aho, "Deep reinforcement based optimization of function splitting in virtualized radio access networks," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021.

[14] Z. Gao, S. Yan, J. Zhang, B. Han, Y. Wang, Y. Xiao, D. Simeonidou, and Y. Ji, "Deep reinforcement learning-based policy for baseband function placement and routing of RAN in 5G and beyond," *Journal of Lightwave Technology*, vol. 40, no. 2, pp. 470–480, 2021.

[15] V. Sciancalepore, L. Zanzi, X. Costa-Perez, and A. Capone, "ONETS: online network slice broker from theory to practice," *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 121–134, 2021.

[16] N. Van Huynh, D. Thai Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1455–1470, 2019.

[17] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. Walid, "Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics," in *IEEE Conference on Network and Service Management (CNSM)*, 2019, pp. 1–5.

[18] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2019.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[20] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[21] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2021.

[22] R. Boutaba, N. Shahriar, M. A. Salahuddin, S. R. Chowdhury, N. Saha, and A. James, "AI-driven closed-loop automation in 5G and beyond mobile networks," in *ACM Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility (FlexNets)*, 2021, p. 1–6.

[23] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

[24] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Springer Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[25] ORAN Alliance, "O-RAN: Towards an open and smart RAN," 2018. [Online]. Available: https://www.o-ran.org/resources

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[28] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

[29] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1609.02907

[30] R. Boutaba, N. Shahriar, M. A. Salahuddin, and N. Limam, *Managing Virtualized Networks and Services with Machine Learning*. Wiley Online Library, 2021.

[31] G. Dandachi, A. De Domenico, D. T. Hoang, and D. Niyato, "An artificial intelligence framework for slice deployment and orchestration in 5G networks," *IEEE Transactions on Cognitive Comm. and Networking*, vol. 6, no. 2, pp. 858–871, Jun. 2020.

[32] J. S. Pujol Roig, D. M. Gutierrez-Estevez, and D. Gündüz, "Management and Orchestration of Virtual Network Functions via Deep Reinforcement Learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 304–317, 2020.

[33] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, K. Samdanis, and X. Costa-Perez, "Optimising 5G infrastructure markets: The business of network slicing," in *IEEE Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.

[34] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, and X. Costa-Pérez, "A machine learning approach to 5G infrastructure market optimization," *IEEE Transactions on Mobile Computing*, vol. 19, no. 3, pp. 498–512, 2019.

[35] H. Yu, F. Musumeci, J. Zhang, Y. Xiao, M. Tornatore, and Y. Ji, "DU/CU placement for C-RAN over optical metro-aggregation networks," in *International IFIP Conference on Optical Network Design and Modeling.* Springer, 2019, pp. 82–93.

[36] Y. Xiao, J. Zhang, and Y. Ji, "Energy-efficient DU-CU deployment and lightpath provisioning for service-oriented 5G metro access/aggregation networks," *Journal of Lightwave Technology*, vol. 39, no. 17, pp. 5347–5361, 2021.

[37] H. Yu, F. Musumeci, J. Zhang, M. Tornatore, and Y. Ji, "Isolation-aware 5G RAN slice mapping over WDM metro-aggregation networks," *Journal of Lightwave Technology*, vol. 38, no. 6, pp. 1125–1137, 2020.

[38] A. Marotta, D. Cassioli, M. Tornatore, Y. Hirota, Y. Awaji, and B. Mukherjee, "Multilayer protection-at-lightpath for reliable slicing with isolation in optical metro-aggregation networks," *Journal of Optical Communications and Networking*, vol. 14, no. 4, pp. 289–302, 2022.

[39] S. Matoussi, I. Fajjari, N. Aitsaadi, and R. Langar, "Deep learning based user slice allocation in 5G radio access networks," in *IEEE Conference on Local Computer Networks (LCN)*, 2020, pp. 286–296.

[40] X. Wang and T. Zhang, "Reinforcement learning based resource allocation for network slicing in 5G C-RAN," in *IEEE Computing, Communications and IoT Applications (ComComAp)*, Shenzhen, China, 2019, pp. 106–111.

[41] S. Matoussi, I. Fajjari, S. Costanzo, N. Aitsaadi, and R. Langar, "5G RAN: Functional split orchestration optimization," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1448–1463, 2020.

[42] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[43] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *arXiv preprint arXiv:1910.07421*, 2019.

[44] K. Poularakis, Q. Qin, F. Le, S. Kompella, and L. Tassiulas, "Generalizable and interpretable deep learning for network congestion prediction," in *IEEE International Conference on Network Protocols (ICNP)*, 2021, pp. 1–10.

[45] P. Zhang, C. Wang, N. Kumar, W. Zhang, and L. Liu, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," *IEEE Internet of Things Journal*, 2021.

[46] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," *IEEE Transactions on Network and Service Management*, 2021.

[47] F. Habibi, M. Dolati, A. Khonsari, and M. Ghaderi, "Accelerating virtual network embedding with graph neural networks," in *IEEE International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–9.

[48] L. M. Larsen, A. Checko, and H. L. Christiansen, "A survey of the functional splits proposed for 5G mobile crosshaul networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 146–172, 2018.

[49] X. Wang, L. Wang, S. E. Elayoubi, A. Conte, B. Mukherjee, and C. Cavdar, "Centralize or distribute? a techno-economic study to design a low-cost cloud radio access network," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[50] Small Cell Forum, "Small cell virtualization functional splits and use cases Rel. 7.0," *Small Cell Forum*, 2016.

[51] C. Desset, B. Debaillie, V. Giannini, A. Fehske, G. Auer, H. Holtkamp, W. Wajda, D. Sabella, F. Richter, M. J. Gonzalez *et al.*, “Flexible power modeling of LTE base stations,” in *IEEE wireless communications and networking conference (WCNC)*, 2012, pp. 2858–2862.

[52] W. Szeto, Y. Iraqi, and R. Boutaba, “A multi-commodity flow based approach to virtual network resource allocation,” in *IEEE GLOBECOM*, vol. 6, 2003, pp. 3004–3008.

[53] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, “On the computational complexity of the virtual network embedding problem,” *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213–220, 2016.

[54] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, “Gradient descent finds global minima of deep neural networks,” 2018. [Online]. Available: https://arxiv.org/abs/1811.03804

[55] S. Huang and S. Ontañón, “A closer look at invalid action masking in policy gradient algorithms,” *arXiv preprint arXiv:2006.14171*, 2020.

[56] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in neural information processing systems*, vol. 30, 2017.

[57] A. Y. Ng, D. Harada, and S. J. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *International Conference on Machine Learning (ICML)*, 1999, p. 278–287.

[58] T. Chen, K. Zhou, K. Duan, W. Zheng, P. Wang, X. Hu, and Z. Wang, “Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study,” 2021. [Online]. Available: https://arxiv.org/abs/2108.10521

[59] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, “RLlib: Abstractions for distributed reinforcement learning,” *arXiv preprint arXiv:1712.09381*, pp. 3053–3062, 2017.

[60] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, “Tune: A research platform for distributed model selection and training,” *arXiv preprint arXiv:1712.09381*, 2018.

[61] J. Parker-Holder, V. Nguyen, and S. Roberts, “Provably efficient online hyperparameter optimization with population-based bandits,” 2020. [Online]. Available: https://arxiv.org/abs/2002.02518

[62] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 1–9.