Multi-agent Distributed Decentralized Dynamic Resource Orchestration in 5G Edge-Cloud Networks

Xingqi Wu†, Junaid Farooq†, and Juntao Chen*
†Department of Electrical and Computer Engineering, University of Michigan-Dearborn, MI USA,
*Department of Computer and Information Science, Fordham University, New York, NY USA,
Emails: {xingqiwu, mjfarooq}@umich.edu, jchen504@fordham.edu.

***Abstract*—Effective resource orchestration for network slicing is critical for optimizing the performance of diverse applications running on next generation communication networks. This paper presents a novel approach that leverages advancements in multi-agent reinforcement learning (MARL) to adaptively learn the resource requirements of various applications in network slices and orchestrate resources in real-time. Our proposed MARL-based orchestration scheme aims to balance the varying requirements of individual network slices, ensuring optimal performance amid dynamic application deployments with limited network information. Simulation results and comparative analyses validate the efficiency and efficacy of our methodology, demonstrating its superiority over traditional methods in terms of system performance and resource utilization. Simulation results indicate that our strategy significantly enhances system utility and efficiency, particularly with limited resources.**

***Index Terms*—edge-cloud network, network slicing, resource orchestration, multi-agent reinforcement learning, traffic flows.**

I. Introduction

Next-generation (NextG) networks are rapidly evolving towards being software-defined and virtualized, enabling unprecedented flexibility in managing the resources of physical network nodes [1], [2]. A key paradigm in these networks is network slicing, which facilitates the creation of virtualized network segments tailored to the specific needs of diverse applications, including industrial automation, remote healthcare, and autonomous vehicles [3]. These applications often demand ultra-reliable low-latency communication (URLLC) and enhanced mobile broadband (eMBB) services, each with distinct performance and resource needs [4].

The architecture of NextG networks comprises of heterogeneous nodes including edge devices, multi-access edge computing (MEC) servers, and core network nodes with varying resource capacities [5]. Each node can provide a proportion of its available resources to the application using it, and hence this local allocation decision can have a significant impact on the overall performance of the application. Applications generate traffic flows, with each flow traversing multiple nodes and links in the network, which can vary in their resource requirements and paths through the network. For example, URLLC traffic flows require low latency and high reliability, whereas eMBB traffic flows require high bandwidth [6]. The end-to-end performance of these flows is critically dependent on the amount of resources allocated by each network node and communication links to the specific types of traffic. Thus, each node and link within the network must continuously manage the allocation of resources to a specific traffic type at any given instant. This dynamic allocation needs to be adaptive to changing network conditions and application requirements, in order to achieve high service quality and resource efficiency from the network providers standpoint. Moreover, the resource provisioning decisions need to be *fine-grained* down to the application flow level.

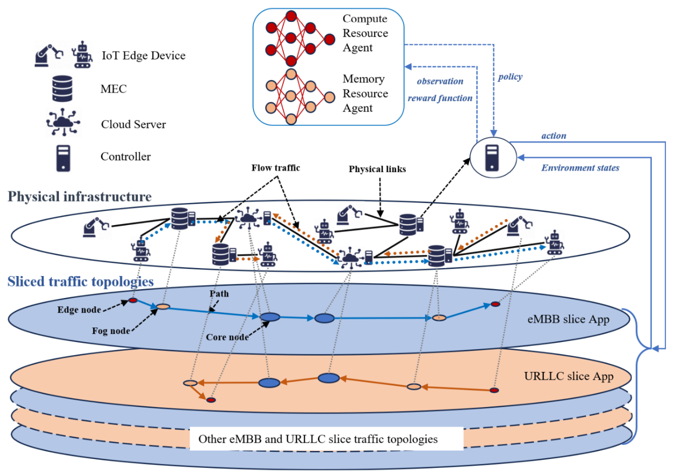


Fig. 1: An abstraction of network slices and traffic flows for applications over an edge-cloud 5G network.

Fig. 1 illustrates the network abstraction showing an example of a sliced traffic flow navigating through an edge-cloud network, comprising of IoT edge devices, MEC servers, and cloud servers. Each flow has specific requirements for computing power and memory. The traffic flows generated by users fluctuate in their requirements over time, leading to uncertain and dynamic resource requests. Access to comprehensive global network information is limited due to rapidly changing environment and the decentralized structure of these distributed resource nodes with network security measures [7]. Therefore, an adaptive and efficient resource orchestration mechanism is critically important to meet end-to-end performance requirements of critical network slices [8]. Traditional static resource allocation methods are inadequate for such dynamic and heterogeneous environments as they fail to capture the intricate and fluctuating traffic dynamics, often leading to sub-optimal resource utilization and an inability to guarantee the necessary service requirements. Therefore, there is a need to actively learn the traffic dynamics in a distributed manner to inform the resource allocation strategy [9].

In this paper, the main objective is to optimize the computational performance of application traffic across the core network, focusing on compute and memory resources[1]. Utilizing an online learning framework, we present a dynamic multi-objective optimization problem considering both slice-averaged efficiency and utility metrics. We then propose a multi-agent reinforcement learning (MARL) based resource orchestration framework [10]. In our approach, each resource provisioning unit is managed by controllers with two types of learning agents, each responsible for a specific resource orchestration policy. These agents can only locally observe data containing flow requirements and slice aggregated resources from the previous time frame. This is especially critical in contexts where resource provisioning units lack visibility into the data of other units and the overall network's physical infrastructure and slice traffic topology. The proposed MARL-based resource orchestration framework is validated through extensive simulations, demonstrating significant improvements over traditional static and random allocation strategies. Our results highlight the potential of MARL to enhance system utility and efficiency in virtualized networks.

The rest of this paper is organized as follows: Section II provides an overview of the related works. Section III introduces the network slicing model and problem formulation. Section IV details the proposed methodology and the MARL-based framework, including the description of actions, observations, reward function, and the training process. Subsequently, Section V describes the simulation results and presents a comparative analysis with baseline schemes. Finally, Section VI concludes the paper.

II. Related Work

Several studies in the literature have focused on virtual network function (VNF) placement and resource provisioning schemes in NextG networks [11]. However, these approaches often assume that functional chains are known prior to resource allocation, which limits their applicability in dynamic environments [12]–[15]. Traditional optimization frameworks have also been explored for resource orchestration, yet their computational complexity and need for repeated execution make them impractical for real-time implementation [16]–[21]. In contrast, recent advancements in artificial intelligence (AI) have introduced dynamic strategies based on deep learning and reinforcement learning for network resource orchestration [22]–[26]. These methods, however, are implemented in radio access networks not considering the complexity of the network topology or typically assume either full access to comprehensive network status and rely on embedding vector exchanges between agents, which may not be feasible in decentralized partially observable environments [27]–[31]. In this work, we address these limitations by introducing

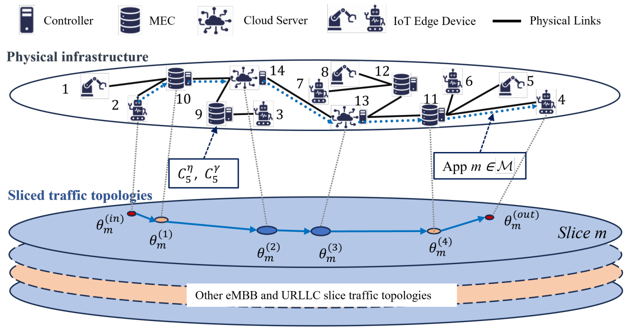


Fig. 2: Illustration of an example application flow in slice m traversing select nodes in the edge-cloud infrastructure.

a MARL-based resource orchestration framework designed for distributed and decentralized network architectures. Our approach enhances system utility and efficiency by dynamically adapting to the stochastic nature of application traffic demands, thereby overcoming the constraints of traditional and existing AI-based methods.

III. System Model

In this section, we construct a network-slicing environment model that depicts a scenario wherein multiple applications concurrently use a shared pool of physical and virtual network resources.

A. Network Infrastructure

We consider a physical network infrastructure consisting of a set of $J_a \in \mathbb{Z}$ access nodes and $J_c \in \mathbb{Z}$ computing nodes (edge and core nodes), denoted by $\mathcal{J} = \{1, 2, \ldots, J\}$, where $J = J_a + J_c$. The system model illustrated in Fig. 2 has 8 access nodes (labeled 1 - 8), 4 edge nodes (labeled 9 - 12), and 2 core nodes (labeled 13 - 14). Each node $j \in \mathcal{J}$ has a fixed compute resource capacity of $C_j^{\eta} \in \mathbb{R}$ and a fixed memory resource capacity of $C_j^{\gamma} \in \mathbb{R}$[2]. These resources are shared among a set of $M \in \mathbb{Z}$ diverse and independent application flows denoted by the set $\mathcal{M} = \{1, 2, \ldots, M\}$. Each application flow $m \in \mathcal{M}$ consisting of multiple functions or microservices traverses a set of nodes in the network.

B. Slice Model

Two categories or slices of application flows are considered in our system, i.e., eMBB and URLLC. The set of application flows of type σ, where $\sigma \in \{1, 2\}$, corresponding to eMBB and URLLC respectively, are denoted by $\mathcal{M}_{\sigma}$, where $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$. Applications in the URLLC slice category have more stringent performance requirements compared to the eMBB slice. Each flow is characterized by the tuple consisting of a starting node, intermediary nodes, and an ending node. Each application flow m initiates at the starting node $\theta_m^{(in)} \in \mathcal{J}$ and terminates at the ending node $\theta_m^{(out)} \in \mathcal{J}$, Hence, an application flow m can be represented by a node set $\Theta_m = \{\theta_m^{(in)}, \theta_m^{(1)}, \theta_m^{(2)}, ..., \theta_m^{(J_m)}, \theta_m^{(out)}\}$, where $\Theta_m \subset \mathcal{J}$ and J_m is the number of intermediary nodes (edge and core nodes)

[1]Several other types of resources can also be considered but these do not significantly alter the proposed resource orchestration scheme.

[2]We assume that the access nodes do not have any compute or memory resources.

traversed by application flow m. We consider a time-slotted system operation where the resource provisioning decisions are undertaken at each time slot t, the duration of which is denoted as T_s. Each application flow $m \in \mathcal{M}$ at time t has compute and memory resource requests denoted by $\mathbf{d}_m^{(t)} = \{\eta_m^{(t)}, \gamma_m^{(t)}\}$. Similarly, each application flow m at time t is allocated with an amount of corresponding resources by node $j \in \Theta_m$ denoted as $\mathbf{r}_{m,j}^{(t)} = \{\zeta_{m,j}^{(t)}, \delta_{m,j}^{(t)}\}$.

C. Utility and Efficiency Characterization

The total allocation of memory resources for application m denoted as $\delta_m^{(t)}$ is calculated by aggregating the resources provided among the resource nodes that the application m traverses, which can be expressed as $\delta_m^{(t)} = \sum_{j\in\Theta_m} \delta_{m,j}^{(t)}$. The utility of memory resource is calculated by $f_\sigma(x)$ defined as:

$$f_\sigma(x) = \begin{cases} \frac{\beta_1}{1+\exp(-(a_1 x + b_1))} + \beta_2 \ln(1+wx), & \sigma = 1, \\ \frac{1}{1+\exp(-(a_2 x + b_2))}, & \sigma = 2, \end{cases} \quad (1)$$

where $x \in \mathbb{R}^+$ indicates the proportion of resource request satisfied by the allocation. For the eMBB slice, the parameters β_1, β_2, a_1, b_1, and w are chosen to ensure that the utility function $f_1(x)$ increases monotonically over the interval $x \in [0, 1]$. This configuration mirrors the eMBB slice's relaxed and flexible requirements. For the URLLC slice, the parameters a_2 and b_2 are set to ensure that the function $f_2(x)$ aligns with the more stringent requirements of the URLLC applications. Insufficient resources result in a drastic decrease in the utility for applications in the URLLC slice, much more than eMBB applications, emphasizing the necessity of resource provisioning to maintain desired utility levels in critical use cases. Moreover, over-provisioning of the memory resource does not provide any additional utility to the application. Thus, we set $f_\sigma(x) = 1$ when $x > 1$. Similarly, we define a monotonically decreasing function of the compute latency, $g_\sigma(z)$, to capture the utility of compute resource, defined as follows:

$$g_\sigma(z) = \frac{1}{(1 + c_\sigma z)}, z > 0, \quad (2)$$

where c_σ is a positive coefficient that determines the slope of the function. At time t, the latency $\tau_m^{(t)}$ of application m can be calculated through a queuing process. Queuing occurs when resource requests are generated before existing requests are processed, potentially leading to increased delays in handling current requests. We consider first-come, first-served policies where requests are processed in the order they arrive [32]. The effective compute resource allocated to the application m calculated based on queued traffic for the compute resource request and the resource allocated to the application is defined as follows:

$$\overset{*}{\zeta}{}_m^{(t)} = \min\left\{\zeta_m^{(t)}, \frac{D_m^{(t-1)}}{T_s} + \eta_m^{(t)}\right\}, \quad (3)$$

where $\zeta_m^{(t)} = \sum_{j\in\Theta_m} \zeta_{m,j}^{(t)}$ is the total amount compute resource allocated to the application flow m. Equation (3) indicates the update of the amount of compute resources that are actually functional on the application m at time t, which signifies that if the compute resources are over-provisioned, it does not provide any additional benefits on application m. Conversely, when the compute resources are allocated insufficiently, the queue of requests start to develop and hence form the residual block. The variable $D_m^{(t)}$ indicates the size of the residual block, which is set to 0 for $t = 0$, and then it is updated each time using

$$D_m^{(t)} = \max\left\{0, D_m^{(t-1)} + T_s\left(\eta_m^{(t)} - \zeta_m^{(t)}\right)\right\}. \quad (4)$$

Therefore, the compute latency can then be determined using the following equation:

$$\tau_m^{(t)} = \frac{2D_m^{(t-1)} - T_s\left(\overset{*}{\zeta}{}_m^{(t)} - \eta_m^{(t)}\right)}{2\zeta_m^{(t)}} + \frac{1}{\overset{*}{\zeta}{}_m^{(t)}}, \quad (5)$$

where the first term represents the queuing delay and the second term indicates processing delays, respectively.

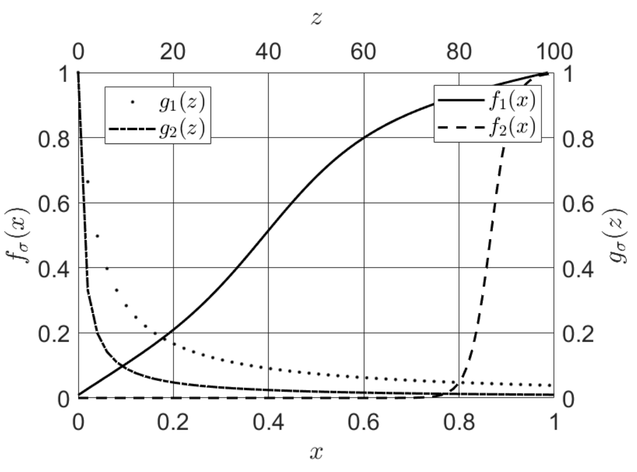


Fig. 3: Utility functions of compute and memory resources

The overall utility of an application flow incorporates the utility of the compute resource and memory resource. The utility function $U_m(\cdot)$ for application m at time t is defined as follows:

$$U_m\left(\mathbf{r}_m^{(t)}\right) = \begin{cases} \alpha_{\eta,1} g_1\left(\tau_m^{(t)}\right) + \alpha_{\gamma,1} f_1\left(\frac{\delta_m^{(t)}}{\gamma_m^{(t)}}\right), m \in \mathcal{M}_1, \\ \alpha_{\eta,2} g_2\left(\tau_m^{(t)}\right) + \alpha_{\gamma,2} f_2\left(\frac{\delta_m^{(t)}}{\gamma_m^{(t)}}\right), m \in \mathcal{M}_2, \end{cases} \quad (6)$$

where the non-negative coefficients $\alpha_{\eta,\sigma}$ and $\alpha_{\gamma,\sigma}$ are chosen for balancing the constituent terms with $\alpha_{\eta,\sigma} + \alpha_{\gamma,\sigma} = 1$. Similarly, an application m in the slice category M_σ at time t has an efficiency of $E_m(\cdot)$, which depends on the utilities of resource provisioning and the amount of resources allocated. The resource efficiency of application flow m can be defined as follows:

$$E_m\left(\mathbf{r}_m^{(t)}\right) = \nu_\eta \frac{g_\sigma\left(\tau^{(t)}\right)}{\zeta_m^{(t)}} + \nu_\gamma \frac{f_\sigma\left(\frac{\delta_m^{(t)}}{\gamma_m^{(t)}}\right)}{\delta_m^{(t)}}, \ m \in \mathcal{M}, \quad (7)$$

where ν_η and ν_γ are non-negative weighting factors with $\nu_\eta + \nu_\gamma = 1$. In the following subsection, we formalize our problem formulation based on these developed metrics of slice utility and resource efficiency.

D. Problem Formulation

The averaged system utility and system efficiency at time t are computed as the combination of the averaged utility and efficiency of the eMBB and the URLLC slices :

$$\overline{U}\left(\mathbf{R}_M^{(t)}, t\right) = \frac{|\mathcal{M}_1|}{|\mathcal{M}|}\overline{U}_1^{(t)}\left(\mathbf{R}_M^{(t)}\right) + \frac{|\mathcal{M}_2|}{|\mathcal{M}|}\overline{U}_2^{(t)}\left(\mathbf{R}_M^{(t)}\right), \quad (8)$$

$$\overline{E}\left(\mathbf{R}_M^{(t)}, t\right) = \frac{|\mathcal{M}_1|}{|\mathcal{M}|}\overline{E}_1^{(t)}\left(\mathbf{R}_M^{(t)}\right) + \frac{|\mathcal{M}_2|}{|\mathcal{M}|}\overline{E}_2^{(t)}\left(\mathbf{R}_M^{(t)}\right), \quad (9)$$

where

$$\overline{U}_\sigma^{(t)}\left(\mathbf{R}_M^{(t)}\right) = \frac{1}{|\mathcal{M}_\sigma|}\sum_{m\in\mathcal{M}_\sigma} U_m\left(\mathbf{r}_m^{(t)}\right), \quad \sigma\in\Sigma, \quad (10)$$

$$\overline{E}_\sigma^{(t)}\left(\mathbf{R}_M^{(t)}\right) = \frac{1}{|\mathcal{M}_\sigma|}\sum_{m\in\mathcal{M}_\sigma} E_m\left(\mathbf{r}_m^{(t)}\right), \quad \sigma\in\Sigma, \quad (11)$$

where $\mathbf{R}_M = \{\mathbf{r}_1, ..., \mathbf{r}_M\}$. The cardinality of $\mathcal{M}$ and $\mathcal{M}_\sigma$ are represented as $|\mathcal{M}|$ and $|\mathcal{M}_\sigma|$ respectively. This measure of system performance encapsulates the relative efficiency and functionality across the respective applications. This formulation can scale to capture a variety of different slices, resource requirements, utility functions, and network topologies. well as utility functions.

Finally, the dynamic multi-objective optimization problem can be mathematically formulated as follows:

$$\max_{\zeta_{m,j}^{(t)},\delta_{m,j}^{(t)}} \left(\frac{1}{T}\sum_{t=0}^{T}\overline{E}\left(\mathbf{R}_M^{(t)}, t\right), \frac{1}{T}\sum_{t=0}^{T}\overline{U}\left(\mathbf{R}_M^{(t)}, t\right)\right)^{\mathrm{T}} \quad (12)$$

$$\text{s.t.} \quad \sum_{m\in\mathcal{M}^j}\zeta_{m,j}^{(t)} \le C_j^\eta, \quad \forall j\in\mathcal{J}, \quad (13)$$

$$\sum_{m\in\mathcal{M}^j}\delta_{m,j}^{(t)} \le C_j^\gamma, \quad \forall j\in\mathcal{J}, \quad (14)$$

where M^j indicates the set of applications that cross the node $j \in \mathcal{J}$, respectively. The objective (12) aims to simultaneously maximize the average system utility and resource efficiency over a time horizon T. The constraints (13) and (14) ensure resources allocated to the application flows at time t by node j cannot exceed the node's compute and memory resource capacity.

IV. Methodology & Framework

This section presents the MARL framework wherein the Dueling Double Deep Q-Network (D3QN) algorithm is implemented to enhance stability and accelerate convergence during the training process.

A. Observations and Actions

We transform the original optimization problem (12)-(14) into a distributed multi-agent problem with an underlying Markov decision process (MDP) defined by the tuple $\left(\mathcal{N}, \mathcal{S}, (\mathcal{O}_i)_{i\in\mathcal{N}}, (\mathcal{A}_i)_{i\in\mathcal{N}}, (R_i)_{i\in\mathcal{N}}, P, \lambda\right)$. The set of all agents is denoted by $\mathcal{N}$ with a cardinality of $|\mathcal{N}| = 2J_c$. The total state space of the environment is denoted by $\mathcal{S}$ and can only be partially observed by each agent, which reflects a degree of information degradation. The action space for agent i is given by $\mathcal{A}_i$, where $\mathcal{A}_1 = \mathcal{A}_2 = ... = \mathcal{A}_{2J_c} = \{1, 2, ..., N_a\}$ with total N_a actions. Matrix P describes the probability of state transitions. The discount factor, λ, captures the decreasing value of future rewards. Furthermore, R_i refers to the reward obtained by agent i following its action, and $o_i \in \mathcal{O}_i$ denotes the observation of agent i, where $\mathcal{O}_i$ is the observation space of agent i.

Considering a type of resource agent associated with the controller of resource node j, the observation of the agent at time t can be denoted as

$$o_{j,h}^t = \Bigg[\sum_{m\in\mathcal{M}_1^j} r_m^{t-1}, \sum_{m\in\mathcal{M}_2^j} r_m^{t-1}, \sum_{m\in\mathcal{M}_1^j} \hat{r}_{m,j}^{t-1}, \sum_{m\in\mathcal{M}_2^j} \hat{r}_{m,j}^{t-1}, \sum_{j\in\Theta_m}\sum_{m\in\mathcal{M}_1^j} \hat{r}_{m,j}^{t-1}, \sum_{j\in\Theta_m}\sum_{m\in\mathcal{M}_2^j} \hat{r}_{m,j}^{t-1}, \Lambda_{j,h}^{t-1}, C_j^h\Bigg], \quad (15)$$

where $h \in \{\eta, \gamma\}$ signifies the index of resource type, while $r_m^t \in \mathbf{d}_m^{(t)}$ and $\hat{r}_{m,j}^t \in \mathbf{r}_{m,j}^{(t)}$ represent the requested and allocated resources respectively. The set of applications in slice σ across the node j is denoted as $\mathcal{M}_\sigma^j$. The variable $\Lambda_{j,h}^{t-1}$ is an array that contains actions on the adaptation of resource levels for each slice category, where $\Lambda_{j,h}^{t-1} = \left[\rho\cdot\phi_1^{t-1}, \rho\cdot\phi_2^{t-1}\right]$, suggesting whether to increase, decrease, or maintain resources in current state. The weighting factor $\rho \in \mathbb{R}$ controls the magnitude of resource increment or reduction. Specifically, $\phi_1^t, \phi_2^t \in \{1, -1, 0\}$, and $\phi_\sigma = 1$ indicates increasing the resource for slice σ, while $\phi_\sigma = -1$ and $\phi_\sigma = 0$ suggest decreasing and maintaining the resource. Therefore, considering two slices in the network, there are three actions of resource allocations for the eMBB slice and three actions for the URLLC slice, which makes 9 potential actions for each resource agent. Given an observation, the agent can select an action out of $N_a = 9$ candidate options represented by $a_{j,h}^t \in \mathcal{A}_i = \{1, 2, ..., 9\}$. Then, the selected action is mapped to the resource allocation decision ϕ_σ^t through a mapping function $\mathcal{F}(\cdot)$ defined as follows:

$$\left(\phi_1^t, \phi_2^t\right) = \mathcal{F}\left(a_{j,h}^t\right). \quad (16)$$

The controller of node j takes $o_{j,h}^{t-1}$ as input and returns the action $a_{j,h}^t$ at time t. Then, the $\hat{r}_{m,j}^t$ can be calculated as follows:

$$\hat{r}_{m,j}^t = \frac{\sum_{m\in\mathcal{M}_\sigma^j}\hat{r}_{m,j}^{t-1} + \rho\cdot\phi_\sigma^t}{|\mathcal{M}_\sigma^j|}, m\in\mathcal{M}_\sigma, j\in\Theta_m. \quad (17)$$

We then incorporate the minimum adjustment method to prevent resource allocation at node j from exceeding the node capacity. This can be described as follows:

$$\hat{r}_{m,j}^{t} \leftarrow \hat{r}_{m,j}^{t} \min \left\{ 1, \frac{C_j^h}{\sum_{m \in \mathcal{M}_1^j \cup \mathcal{M}_2^j} \hat{r}_{m,j}^{t}} \right\}. \tag{18}$$

B. Reward Function

The performance of our proposed resource orchestration framework is based on the optimal policies of the agents, which are designed to maximize the system's overall utility and resource efficiency. Thus, given a resource agent associated with the controller in node j and an application set $\mathcal{M}_1^j \cup \mathcal{M}_2^j$, the reward at time t is formulated as follows:

$$\begin{aligned} R_{j,h}^t(o_{j,h}^t, a_{j,h}^t) = &-B_1 \cdot F\left(\sum_{j \in \Theta_m} \sum_{m \in \mathcal{M}_1^j} \hat{r}_{m,j}^t - \sum_{m \in \mathcal{M}_1^j} r_m^t \right) \\ &- B_2 \cdot F\left(\sum_{j \in \Theta_m} \sum_{m \in \mathcal{M}_2^j} \hat{r}_{m,j}^t - \sum_{m \in \mathcal{M}_2^j} r_m^t \right) + B_3, \end{aligned} \tag{19}$$

where B_1, B_2, and B_3 are positive weighting parameters to balance the contribution of each term within the reward function. The value of B_2 should far exceed B_1, reflecting the priority of the URLLC slice. The function $F(\cdot)$ is defined as follows:

$$F(x) = \begin{cases} y(x), & x < 0, \\ 0, & \text{else}, \end{cases} \tag{20}$$

where $y(x)$ is a convex, monotonically decreasing function with $y(x) = 0$ at $x = 0$. The reward function is a compound measure that amalgamates two distinct square terms: the former aggregates the difference between the total amount of a particular type of resource h allocated to the eMBB application flows at node j and the aggregated resource request of the eMBB application flows at time t. Similarly, the latter term indicates the same process for the flows in the URLLC slice category across node j at time t. Consequently, the reward function at node j exhibits interdependence between multiple agents.

C. Training Procedure

We use a Markov chain (MC) to generate the requirements for each application flow $m \in \mathcal{M}_\sigma$. The MC model has N_s states which are randomly sampled within a range specified in Table I. The transition matrix $\mathbf{P}$ is constrained such that $\mathbf{P}_{i,j} = 0 : |i - j| > 1, \forall\ i, j \in \{1, 2, ..., N_s\}$. This implies that state transitions are restricted to neighboring states only based on the given probabilities. Each application flow has an independent sampling process from the N_s requirement states. The variability in the number of application flows and their respective resource requests can aggregate to exceed network capacity, making selective resource allocation necessary. During the training, the policies of the agents may cause some application flows to suffer reduced performance as the system endeavors to maximize overall system performance. An appropriate level of aggregation can help increase the dynamics of the network environment, enabling agents to learn the optimal policy effectively. At the beginning of every episode, we generate a random number of application flows, each assigned a static route consisting of a randomly generated node path. Each episode has T time slots. We update resource requests for every episode (every time slot during testing) based on the transition matrix $\mathbf{P}$. As training proceeds, we gradually decrease the exploration to encourage the agent to utilize its learned strategies, preventing it from settling for sub-optimal actions.

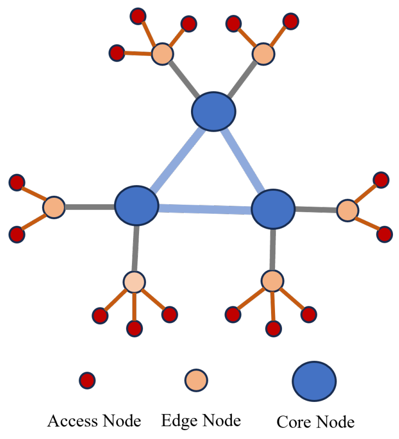


Fig. 4: Example network topology comprising of access nodes, edge nodes, and Core nodes. The size of the nodes are proportional to the amount of compute and memory resources available at those nodes.

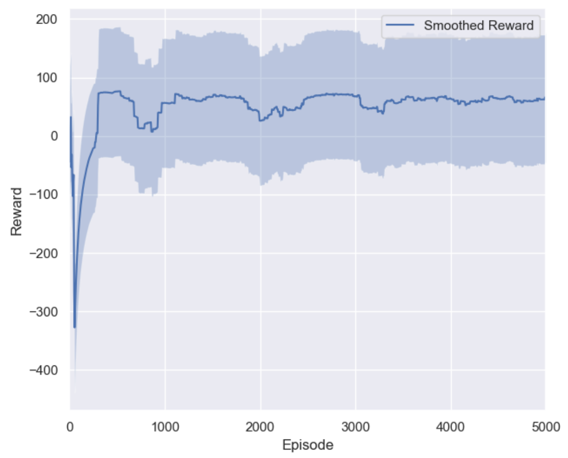


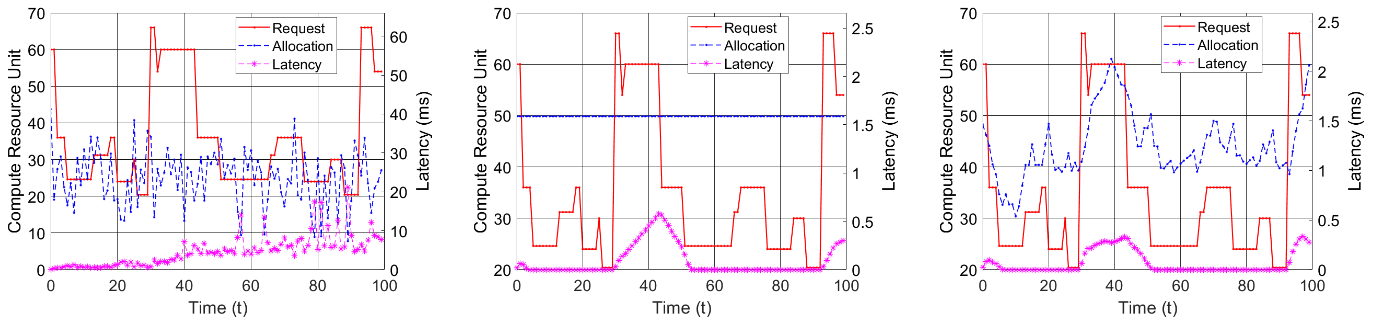
Fig. 5: Evolution of averaged training reward for compute resource agents.

V. Simulation and Results Analysis

In this section, we first describe the simulation settings and subsequently present the results for evaluating the performance of our proposed framework and comparing it with baseline strategies.

A. Simulation Setup

Our simulations are conducted using a Python 3.7 environment, utilizing the Pytorch 1.10.1 library. The training



(a) Random Strategy (b) Static Strategy (c) MARL

Fig. 6: Compute resource allocation for an application of URLLC slice category.

Parameters	Value
Number of application traffic chains, $\mathcal{M}$	2∼8
Compute resource capacity at node j, C_j^η	$\{30, 60\}$
Memory resource capacity at node j, C_j^γ	$\{30, 60\}$
Compute resource demand of App m, η_m	$20 \sim 65$
Memory resource demand of App m, γ_m	$30 \sim 75$
Total training episodes N_{train}	5000
Total testing episodes N_{test}	25
Number of MDP states N_s	10
Learning rate, μ	5e-5
Discount factor, λ	0.99
Time segments for each episode, T	100
Function $f_1(\cdot)$ parameters (β_1, β_2, a_1, b_1, w)	0.2, 0.8, 10, -4, 1.8
Function $f_2(\cdot)$ parameters (a_2, b_2)	45, -40
Function $g_\sigma(\cdot)$ parameters (c_1, c_2)	0.2, 1
Slice 1 Utility weight parameters ($\alpha_{\eta,1}$, $\alpha_{\gamma,1}$)	0.5, 0.5
Slice 2 Utility weight parameters ($\alpha_{\eta,2}$, $\alpha_{\gamma,2}$)	0.5, 0.5
Efficiency weight parameters (ν_η, ν_γ)	0.5, 0.5
Resource node number, N	9
Relay buffer size	5e5
Network soft update parameter	5e-3
Amplitude of guidance, ρ	3
Performance function constants (B_1, B_2, B_3)	0.05, 5, 100
Hidden network layer size	[25, 25]
Observation Size	9
Action Size	1
Batch Size	256

TABLE I: Simulation Parameters.

was carried out on an Nvidia GeForce MX350 with 8 GB memory and took 23 hours. The network topology used in our simulation experiments is shown in Fig. 4, which is inspired by the edge-cloud architecture in next generation communication applications particularly in industrial settings [33]. The network topology contains three core nodes, each connected to two edge nodes, each having distinct levels of compute and memory resource capacities. Each edge node is interconnected with a multitude of access nodes. Access nodes serve as critical junctures, acting as the start and end points for flow traffic. During the simulation, we construct application flows by initial selection of a pair of access nodes. Subsequently, the intermediate paths are determined by randomly sampling from the available paths between the two nodes. The nodes within any given flow path must be unique, thereby excluding the possibility of cyclic paths. During the training phase, the initial resource capacities for both the edge nodes and the core nodes were configured according to normal distributions,($\mu = 30, \sigma^2 = 1$) for the edge nodes and ($\mu = 60, \sigma^2 = 2$) for the core nodes. We choose $y(x) = x^2$ for reward function $R_{j,h}^t(\cdot)$ in equation (20). A comprehensive list of parameter values used in the simulations is provided in Table I.

B. Performance Analysis

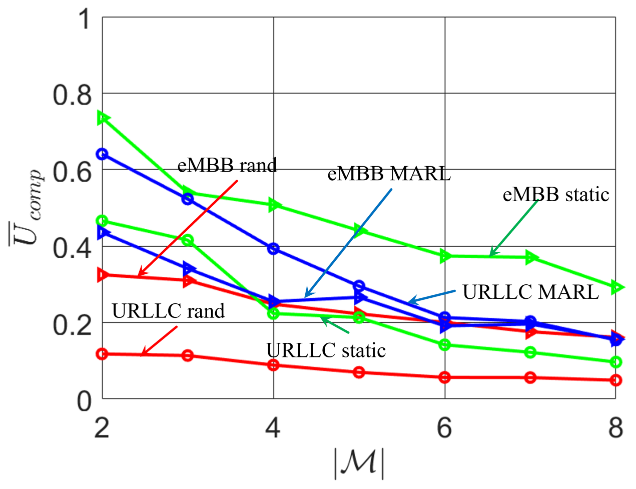
The averaged training results of all agents across the network controlling a particular resource are depicted in Fig. 5, which illustrates the convergence trend of the learning process over the episodes.

We use the following baseline methods as benchmarks for comparative analysis:

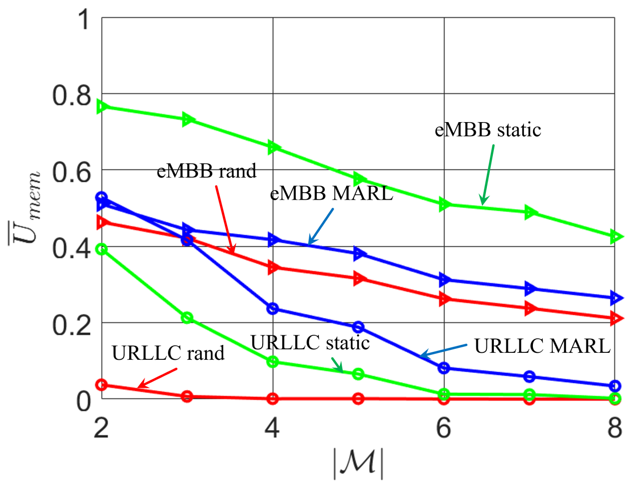
1) **Random Strategy:** Allocation of resources to each slice at every resource node was executed in a stochastic manner without regard to the specific needs or priorities of the slices.
2) **Static Strategy:** A predetermined static allocation protocol that each slice receives an equal, invariant allocation of fifty percent of the total resource capacity at every node, regardless of the fluctuating aggregated requirements of the individual slices.

Fig. 6 exhibits the requested, allocated compute resource, and delay patterns observed under different orchestration approaches, providing an intuitive understanding of how the orchestration of compute resource affects compute latency. In Fig. 6a, the allocation pattern resulting from the random strategy is displayed. It is evident that resource allocation under this strategy does not correlate with the pattern of resource requirements. Fig. 6b shows the allocation patterns that emerge from the static strategy. From this observation, the allocation remains fixed and fails to adapt to the dynamic nature of resource requests over time. As demonstrated in Fig. 6c, the DRL strategy showcases an adaptive and responsive pattern of orchestration. This strategy dynamically reduces resource allocation when the demands are fulfilled and increases the allocation in response to unmet requests.

In the subsequent analysis, we examined the system utility and efficiency considering both compute and memory resource orchestration given the limited resource capacity. For both

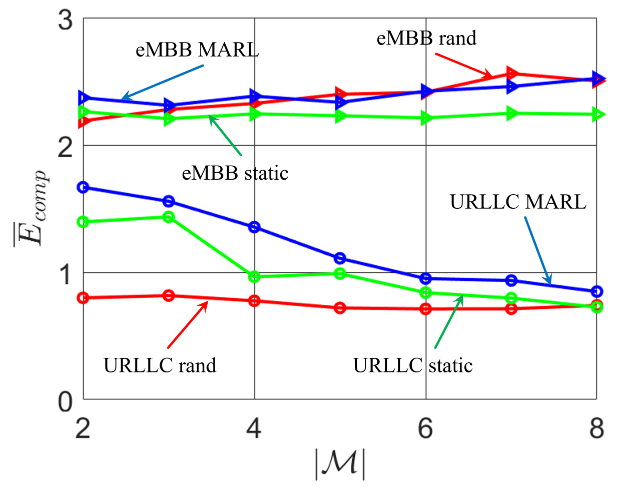


(a) Slice compute resource utility

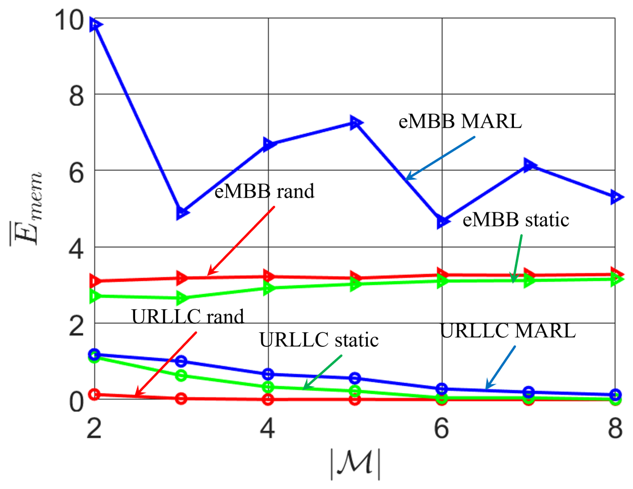


(b) Slice memory resource utility

Fig. 7: Averaged efficiency vs the number of flows.



(a) Slice compute resource efficiency



(b) Slice memory resource efficiency

Fig. 8: Averaged efficiency vs the number of flows.

types of resources, the core nodes are provisioned with an overall capacity of 20 units, while the fog nodes have a designated capacity of 10 units. The methodology for defining utility and efficiency metrics is presented in Section III. We initiated the process by varying the total number of application flows from 2 to 8 incorporated in the system. For each flow number setting, we undertook N_{test} episodic tests to evaluate performance. The results obtained from these episodes were then averaged, providing a measure of efficiency and utility under the various orchestration approaches. Fig. 7 reveals that the static method achieves superior utility specifically for eMBB slice. Conversely, the random strategy demonstrates the lowest utility for both slices. However, it's significant to point out that the DRL strategy delivers the highest utility for the URLLC slice. Considering the system orchestration efficiency, as illustrated in Fig. 8, it is worth noting that the DRL strategy demonstrates enhanced resource efficiency for all types of slices. Specifically, for the eMBB slice, it is evident that a smaller resource allocation achieves greater utility, ascribed to the comparatively relaxed requirements of applications within this slice. In contrast, the demanding characteristics of the URLLC slice necessitates a greater allocation of resources to satisfy its stringent requirements.

VI. Conclusion

In our study, a thorough network slicing environment model has been developed, accounting for two categories of slices. We propose a resource orchestration scheme that employs a multi-agent reinforcement learning (MARL) framework to manage the dynamic and diverse network conditions inherent to decentralized network architectures. Extensive comparative evaluations have confirmed the enhanced system utility and efficiency of our MARL-based resource orchestration approach, under conditions of scarce resources. For future work, we plan to augment our network resource orchestration framework with an additional class of agent, specifically designed for resource reservation. By integrating this new agent, we anticipate the establishment of more robust defensive strategies that will maintain the integrity and reliability of network services under adverse conditions.

References

[1] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3298–3304, 2018.

[2] M. Chahbar, G. Diaz, A. Dandoush, C. Cérin, and K. Ghoumid, "A comprehensive survey on the E2E 5G network slicing model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 49–62, 2021.

[3] S. Wijethilaka and M. Liyanage, "Survey on network slicing for Internet of things realization in 5G networks," *IEEE Communications Surveys Tutorials*, vol. 23, no. 2, pp. 957–994, 2021.

[4] R. Khan, P. Kumar, D. N. K. Jayakody, and M. Liyanage, "A survey on security and privacy of 5G technologies: Potential solutions, recent advancements, and future directions," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 196–248, 2020.

[5] X. Wu, J. Farooq, and J. Chen, "Adaptive risk-aware resource orchestration for 5G microservices over multi-tier edge-cloud systems," in *IEEE International Conference on Communications Workshops (ICC Workshops 2024)*, Denver CO, USA, Jun. 2024.

[6] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[7] C. De Alwis, P. Porambage, K. Dev, T. R. Gadekallu, and M. Liyanage, "A survey on network slicing security: Attacks, challenges, solutions and research directions," *IEEE Communications Surveys Tutorials*, vol. 26, no. 1, pp. 534–570, 2024.

[8] S. D. A. Shah, M. A. Gregory, and S. Li, "Cloud-native network slicing using software defined networking based multi-access edge computing: A survey," *IEEE Access*, vol. 9, pp. 10 903–10 924, 2021.

[9] H. P. Phyu, D. Naboulsi, and R. Stanica, "Machine learning in network slicing—a survey," *IEEE Access*, vol. 11, pp. 39 123–39 153, 2023.

[10] K. Zhang, Z. Yang, and T. Başar, *Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms*. Cham: Springer International Publishing, 2021, pp. 321–384. [Online]. Available: https://doi.org/10.1007/978-3-030-60990-0_12

[11] A. Ghosh, A. Maeder, M. Baker, and D. Chandramouli, "5G evolution: A view on 5G cellular technology beyond 3GPP release 15," *IEEE Access*, vol. 7, pp. 127 639–127 651, 2019.

[12] H. U. Adoga and D. P. Pezaros, "Towards latency-aware vNF placement on heterogeneous hosts at the network edge," in *IEEE Global Communications Conference (Globecom 2023)*, Kuala Lumpur, Malaysia, 2023, pp. 6383–6388.

[13] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal VNFs placement in CDN slicing over multi-cloud environment," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 616–627, 2018.

[14] D. Gedia and L. Perigo, "Decision-tree placement algorithm for containerized voip VNFs: A network management approach," in *25th Conference on Innovation in Clouds, Internet and Networks (ICIN 2022)*, Paris, France, 2022, pp. 1–5.

[15] D. M. Manias, A. Chouman, J. Naoum-Sawaya, and A. Shami, "Resilient and robust QoS-preserving post-fault VNF placement," *IEEE Networking Letters*, vol. 5, no. 4, pp. 270–274, 2023.

[16] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1277–1287, 2021.

[17] C. Gao, A. Shaan, and A. Easwaran, "Deadline-constrained multi-resource task mapping and allocation for edge-cloud systems," in *IEEE Global Communications Conference (Globecom 2022)*, Rio de Janeiro, Brazil, 2022.

[18] C. Gao and A. Easwaran, "Work-in-progress: Deadline-constrained multi-resource allocation in edge-cloud system," in *IEEE Real-Time Systems Symposium (RTSS 2022)*, Houston, TX, USA, 2022, pp. 503–506.

[19] J. Lee, H. Ko, J. Kim, and S. Pack, "DATA: Dependency-aware task allocation scheme in distributed edge clouds," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7782–7790, 2020.

[20] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung, "Distributed resource allocation and computation offloading in fog and cloud networks with non-orthogonal multiple access," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 137–12 151, 2018.

[21] H. Halabian, "Distributed resource allocation optimization in 5G virtualized networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 627–642, 2019.

[22] S. Jiang, J. Zheng, F. Yan, and S. Zhao, "Reinforcement-learning-based network slicing and resource allocation for multi-access edge computing networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 10, no. 3, pp. 1132–1145, 2024.

[23] K. Suh, S. Kim, Y. Ahn, S. Kim, H. Ju, and B. Shim, "Deep reinforcement learning-based network slicing for beyond 5G," *IEEE Access*, vol. 10, pp. 7384–7395, 2022.

[24] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, "Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach," *IEEE Open Journal of Vehicular Technology*, vol. 2, pp. 272–288, 2021.

[25] B. Shang, L. Liu, and Z. Tian, "Deep learning-assisted energy-efficient task offloading in vehicular edge computing systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9619–9624, 2021.

[26] D. Ayepah-Mensah, G. Sun, G. O. Boateng, S. Anokye, and G. Liu, "Blockchain-enabled federated learning-based resource allocation and trading for network slicing in 5G," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 654–669, 2024.

[27] X. Cheng, Y. Wu, G. Min, A. Y. Zomaya, and X. Fang, "Safeguard network slicing in 5G: A learning augmented optimization approach," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1600–1613, 2020.

[28] F. Mason, G. Nencioni, and A. Zanella, "Using distributed reinforcement learning for resource orchestration in a network slicing scenario," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 88–102, 2023.

[29] Y. Kim and H. Lim, "Multi-agent reinforcement learning-based resource management for end-to-end network slicing," *IEEE Access*, vol. 9, pp. 56 178–56 190, 2021.

[30] P. Doanis and T. Spyropoulos, "Sample-efficient multi-agent DQNs for scalable multi-domain 5G+ inter-slice orchestration," *IEEE Transactions on Machine Learning in Communications and Networking*, vol. 2, pp. 956–977, 2024.

[31] W. Wang, L. Tang, T. Liu, X. He, C. Liang, and Q. Chen, "Toward reliability-enhanced, delay-guaranteed dynamic network slicing: A multiagent DQN approach with an action space reduction strategy," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 9282–9297, 2024.

[32] A. Brandwajn and T. Begin, "First-come-first-served queues with multiple servers and customer classes," *Performance Evaluation*, vol. 130, pp. 51–63, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166531618300701

[33] T. N. Alrumaih and M. J. Alenazi, "GENIND: An industrial network topology generator," *Alexandria Engineering Journal*, vol. 79, pp. 56–71, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S111001682300649X