AFEDA : Enhancing Network Slices Acceptance Ratio with Transformer-based Feature Extraction

Abdel Kader Chabi Sika Boni, Hassan Hassan, Khalil Drira

AFEDA : Enhancing Network Slices Acceptance Ratio with Transformer-based Feature Extraction

Abdel Kader Chabi Sika Boni
LAAS-CNRS
Université de Toulouse, CNRS, UPS
Toulouse, France
akchabisik@laas.fr

Hassan Hassan
LAAS-CNRS
Université de Toulouse, CNRS, UPS
Toulouse, France
hhassan@laas.fr

Khalil Drira
LAAS-CNRS
Université de Toulouse, CNRS, UPS
Toulouse, France
khalil@laas.fr

***Abstract*—The advent of 5G network slicing technology makes it possible to divide a shared infrastructure into logical networks called network slices, each providing a customized quality of service (QoS). Consequently, the QoS satisfaction has shifted from "how to provide tailored QoS?" to "how to improve the acceptance ratio of deployed network slices?". Existing works have successfully employed Deep Reinforcement Learning (DRL) agents to address this challenge, proposing various placement strategies guided by reward functions. In this paper, we introduce AFEDA, an improved DRL agent by coupling it with a transformer-based active features extractor and demonstrate that beyond the reward function, extracting active features from observations significantly helps a DRL agent to enhance the acceptance ratio of slices. Through extensive simulations on two infrastructures, we show that AFEDA is capable of placing 14% to 31% more slices compared against a DRL agent with same configurations while using the same observations as raw features.**

***Keywords*—Network slicing, Deep reinforcement learning, Transformer, Quality of Service.**

I. Introduction

The proper functioning of critical applications and services is subject to good quality of service (QoS). For example, Internet of Things (IoT) services evolved, demanding diverse, varied, and sometimes conflicting QoS. By 2025, projections suggest that the global count of IoT devices could range anywhere from 25 billion to as high as 100 billion [1] [2] while traditional QoS management mechanisms are reaching their limits in satisfying such requirements. Recognizing this issue, the Next Generation Mobile Network (NGMN) proposed the concept of Network Slicing in 5G networks [3].

Network slicing refers to a technique that partitions a shared physical infrastructure into multiple isolated virtual networks known as slices, each engineered to deliver distinct performance levels and service attributes according to predefined Quality of Service (QoS) requirements. The concept is based on two technologies, namely Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) [4]. The main challenge with Network Slicing is how to optimally manage the resources of the physical infrastructure to deploy a maximum number of slices.

Among different optimization techniques, Deep Reinforcement Learning (DRL) has been used successfully in various domains such as object detection, video games, self-driving cars, etc [5] to achieve autonomous learning behavior. In DRL, an agent interacts with an environment through an observation-action-reward mechanism. DRL agents ability to self-improve their learned policy based on reward feedback led to many works employing them to solve the network slicing problem. The approach is iterative with DRL agents receiving joint description of the physical infrastructure and slice requests as observations and proposing computing nodes in the infrastructure on which to place each Virtual Network Function (VNF) in the slice requests.

Throughout this study, network slices are denoted by the term Network Slice Placement Request (NSPR). We extend a DRL algorithm specifically a Double Deep Q-Networks (DDQN) agent with a Transformer-based active features extractor, named it **AFEDA** (**A**ctive **F**eatures **E**xtractor in **D**rl **A**gent). And we experimentally prove that even with the same reward function, the same fully connected neural network structure, the same slicing infrastructures and the same environment conditions, a DDQN agent extracting active features from observations (like AFEDA does) achieves significant increase in NSPRs acceptance ratio (up to 14% to 31% more NSPRs placed) than a DDQN agent processing the same observations as ready-to-go features as done in existing works.

The contributions of this work are outlined as follows:

- we propose AFEDA, an improved DDQN agent in which we repurpose the transformer architecture as an active features extractor in the context of network slicing
- we compare AFEDA against a DDQN agent with same hyperparameters, same rewards and environment conditions to highlight the impact of the active features extractor
- we conduct our simulations with two infrastructures from literature to add credibility to our simulations and conclusions

The remainder of this paper is structured as follows. In Section

II, we delve into various approaches taken by related works that have addressed the problem of network slicing optimization. We provide the network slicing problem statement in Section III while we explain how AFEDA works and interacts with its environment in Section IV. Our fine-tuning and comparison results are presented in Section V. Finally, we conclude and provide our coming extended work plans in Section VI.

II. Related work

The work in [6] employs a Proximal Policy Optimization (PPO) agent to solve a Radio Access Network (RAN) slicing optimization problem, aiming to maximize Infrastructure Provider (InP) revenue by selecting profitable slices and minimizing server usage. [7], [8], [9] and [10] adopted similar approach while optimizing latency, reliability, and availability [7], minimizing resource utilization using a Deep Q-Network (DQN) agent [8], optimizing resource allocation and traffic routing through Graph Convolutional Network (GCN)-powered dueling DQN [9] or maximizing the long-term utility of network slices using a Twin Delayed Deep Deterministic (TD3) agent [10].

The authors in [11] succeed in improving the cost-efficiency and the acceptance ratio of network slices by employing a DDQN agent to determine VNFs placement, Dijkstra algorithm for finding shortest paths between VNFs, and a Binary Search Assisted Gradient Descent (BSAGD) for cost-effective service realization. In [12], three Asynchronous Actor Critic (A2C) agents are used to generate three continuous values representing the amount of Central Processing Unit (CPU), Random Access Memory (RAM) and storage to allocate to slices. [13] showcased a multi-environment framework where 24 GCN-powered Asynchronous Advantage Actor Critic (A3C) agents independently perform slices placement. [14] proposed DeepViNE to deploy grid-like slice requests on grid-like infrastructures. In their proposal, an initial deployment is assigned to the slice request which is then moved iteratively through grid based actions (move top, down, right and left) until its final position. Authors in [15] and [16] employ an Epochal Stochastic Bandit Algorithm Selection (ESBAS) to select iteratively the best agent among three Deep Deterministic Policy Gradient (DDPG) aiming at maximizing slices acceptance ratio. NFVdeep is proposed in [17] to jointly place and minimize the operating cost of the occupied nodes in the infrastructure as well as the total throughput of accepted slice requests. With RDAM, [18] significantly increases the long-term average revenue, the long-term revenue-to-cost ratio and the acceptance ratio.

The works in [19] and [20] introduced HA-DRL in which they used heuristic known as the "Power of Two Choices" was employed to guide an A3C agent enhanced by a Graph Convolutional Network (GCN). The authors support their approach by showing that the heuristic accelerates the agent's initial exploration phase. Their research is further expanded in [21], addressing dynamic network environments and adapting to realistic traffic load scenarios [22] showing that the use of a heuristic can improve a DRL agent performance, even if those heuristics methods may converge prematurely to suboptimal solutions, remaining trapped in a local optimum that is significantly distant from the global best outcome [23] [24].

In all these previous works, the improvement of the acceptance ratio of network slices was achieved by emphasizing the selection of a DRL agent deemed efficient by the authors, combined with a reward function. Our paper takes a different perspective, demonstrating that even with the same DRL agent, reward function, and DRL environment conditions, employing active features extraction significantly enhances the acceptance ratio of network slices. Finally, among the articles referenced in this section DDQN and its less stable version, DQN, were used in more than half of the papers. So we chose to implement AFEDA on top of DDQN because the proposed improvement would indirectly cover a large majority of the works addressing network slicing using DRL.

III. Network slicing problem statement

A. Physical Infrastructure

Let $n_i \in N,\ i \in [1, |N|]$ be any node in the physical network N, comprising computing nodes, routers, and switches. Let $C \subseteq N$ denote the subset of computing nodes able to host VNFs, with $c_i \in C,\ i \in [1, |C|]$.

1) Computing Nodes: Each node $c_i \in C$ is defined by its available resource capacities: CPU ($\mathcal{R}_i^{cpu}$), RAM ($\mathcal{R}_i^{ram}$), and storage ($\mathcal{R}_i^{stor}$).

2) Physical Links: A physical link $(n_i, n_{i'}),\ i \neq i',\ i, i' \in [1, |N|]$ connects either: (i) a computing node to an SDN switch, (ii) an SDN switch to a router, or (iii) two routers. Each link has an available bandwidth $\mathcal{BW}_{(n_i, n_{i'})}$.

3) Node Bandwidth: The bandwidth of node $c_i \in C$ is given by the sum of available bandwidths of adjacent links: $\mathcal{BW}^i = \sum \mathcal{BW}_{(c_i, n_{i'})}$, where $n_{i'}$ is a neighbor of c_i.

B. Network Slice Placement Request

The structure of an NSPR is shown in Figure 1. It consists of a set P of VNFs connected via virtual links.

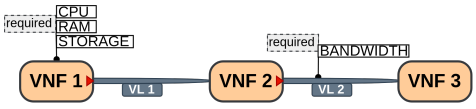


Fig. 1: Example of NSPR with $|P| = 3$ VNFs each requiring CPU, RAM, and storage resources. Consecutive VNFs are linked by a virtual link (VL) that requires a specific bandwidth resource.

1) VNF p_j description: Each $p_j \in P,\ j \in [1, |P|]$ has resource demands: CPU ($\mathcal{U}_j^{cpu}$), RAM ($\mathcal{U}_j^{ram}$), and storage ($\mathcal{U}_j^{stor}$).

2) Virtual Link Description: The required bandwidth for the virtual link $(p_j, p_{j'}),\ j' = j + 1$ is denoted $bw_{(p_j, p_{j'})}$.

3) VNF Bandwidth: A VNF p_j has bandwidth $bw^j = bw_{(p_{j'}, p_j)}$ with $j' = j - 1$. The first VNF has no incoming link, hence $bw^1 = 0$.

C. Remaining Resources

Post-placement of p_j on c_i, the remaining resources are: CPU $\overline{\mathcal{R}}_{j,i}^{cpu}$, RAM $\overline{\mathcal{R}}_{j,i}^{ram}$, and storage $\overline{\mathcal{R}}_{j,i}^{stor}$. Thus, $\overline{\mathcal{R}}_{j,i}^{cpu} = \mathcal{R}_i^{cpu} - \mathcal{U}_j^{cpu}$, $\overline{\mathcal{R}}_{j,i}^{ram} = \mathcal{R}_i^{ram} - \mathcal{U}_j^{ram}$ and $\overline{\mathcal{R}}_{j,i}^{stor} = \mathcal{R}_i^{stor} - \mathcal{U}_j^{stor}$. n the other hand, after satisfying the bandwidth requirement $bw_{(p_j,p_{j'})}$ of a virtual link $(p_j, p_{j'})$, the remaining bandwidth of the physical link $(n_i, n_{i'})$ is defined as $\overline{\mathcal{BW}}_{(n_i,n_{i'})/(p_j,p_{j'})} = \mathcal{BW}_{(n_i,n_{i'})} - bw_{(p_j,p_{j'})}$.

D. Successful VNF and NSPR

A VNF p_j is successfully placed on a computing node c_i when $\overline{\mathcal{R}}_{j,i}^{cpu} \geq 0$, $\overline{\mathcal{R}}_{j,i}^{ram} \geq 0$, $\overline{\mathcal{R}}_{j,i}^{ram} \geq 0$ and $\overline{\mathcal{BW}}_{l/(p_{j-1},p_j)} \geq 0, \forall l \in \mathcal{L}$. The latter condition means any physical link l in physical path $\mathcal{L}$ linking the computing node hosting VNF p_j to the one hosting VNF p_{j-1} must satisfy the minimum bandwidth required by virtual link (p_{j-1}, p_j). Concerning the chaining process, we get inspired by [25] and [11] to use a constrained Breadth-First Search (BFS) algorithm to perform the virtual links matching.

An NSPR is successful when all its VNFs are successfully placed.

E. NSPR Placement formalization

The slice placement challenge can be defined as follows: i) an instance of a NSPR specifies a set of VNFs and VLs with their respective requirements; ii) a substrate network is provided, composed of physical nodes and links with finite resource capacities; iii) the objective is to determine an efficient mapping of each VNF to a suitable physical node and each VL to an appropriate path within the underlying infrastructure. This formulation approach has been extensively adopted in the literature, such as in [19].

IV. Active Features Extractor in DRL Agent

AFEDA interacts with its environment through an observation-action-reward mechanism as visible in Figure 2.

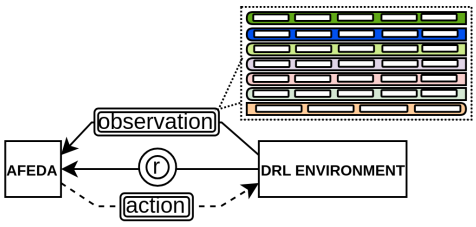


Fig. 2: Overview of an interaction between AFEDA and its environment. The latter contains the infrastructure and slice requests.

The environment contains both the physical infrastructure and received NSPRs. Independently, each NSPR's VNFs are retrieved and processed iteratively. The input observation takes the form of a matrix composed of two main components: 1) the available resource capacities (CPU, RAM, storage, and bandwidth) across all physical computing nodes, represented in the first $|C|$ rows, and 2) the resource demands of the VNF currently under consideration, captured in the final row. Upon receiving this observation, AFEDA selects an action, which corresponds to the identifier of the physical node where the current VNF is to be allocated.

A. Action generation

AFEDA's internal process of action generation is visible in Figure 3. First, the observation goes through AFEDA's active features extractor. A sequence of token-like vectors (element **(A)** in Figure 3) is built by concatenating three set of values for each computing node i) CPU, RAM and storage resources, ii) the bandwidth of the computing node that hosts the most recently deployed VNF of the NSPR and iii) the required CPU, RAM and storage resources by the VNF. That step is important to comply with the token notion as we use the encoder part (element **(B)** in Figure 3) of the Transformer architecture proposed in [26] as the core element of our active features extractor. AFEDA passes the sequence through stacked encoders to get the resulting active features (element **(C)** in Figure 3) which are input to the DDQN algorithm after a flattening operation. We hypothesize the stacked encoders to have two roles : 1) the dynamic attention scores assigned to each computing node's token inside the multi-head attention would help AFEDA to dynamically prioritize the nodes which are likely to enhance the overall slices acceptance ratio and 2) the unicity of the extracted active features would foster AFEDA to define a specific policy for each observation instead of applying the classic DRL agents' similarity policy where two similar observations systematically yield the same action.

To provide its action i.e, the index of the physical computing node on which to deploy the ongoing VNF, AFEDA inputs the flattened active features into the fully connected policy neural network (element **(D)** in Figure 3) resulting in a vector of Q-values from which the action is sampled using the argmax operator. Note that the Q-value of an action is an estimation of the cumulative discounted reward expected when choosing that action. Q-values are updated during the training process as AFEDA interacts with its environment..

B. Training process

After each interaction with the environment as depicted in Figure 2, AFEDA stores the observation o_t, the action a_t, the reward r_t along with the next observation $o_t^{'}$ in its prioritized replay memory (element **(E)** in Figure 3) as an experience. Periodically, a sample of experiences are retrieved to train the policy network by backpropagating the mean square error between observation o_t's Q-values and the desired Q-values obtained through Equation (1)

$$Y^{DDQN} = r_t + \gamma Q_{target}(o_t^{'}, \underbrace{\text{argmax}}_{a_t} Q_{policy}(o_t^{'}, a)) \quad (1)$$

where Q_{target} and Q_{policy} represent the target and evaluation neural networks respectively while $\gamma \in [0, 1]$ represents the discount rate, which regulates the significance attributed to anticipated future rewards in the computing of Y^{DDQN}.

The stacked encoders are trained during the same back-propagation eliminating the requirement to train the active features extractor separately. The training process ensures the

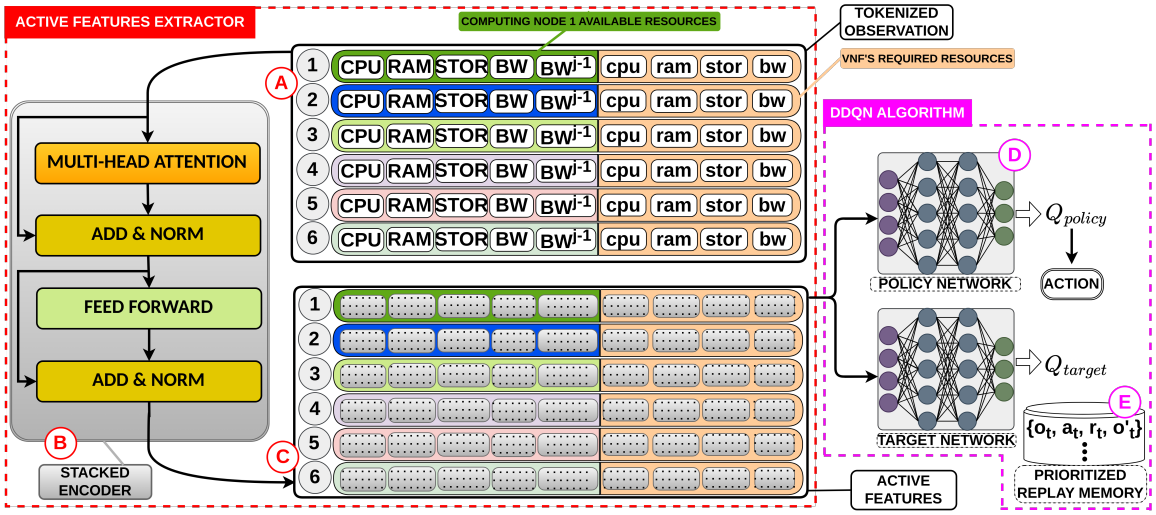


Fig. 3: AFEDA's overview considering a slicing infrastructure of 6 computing nodes. For the sake of simplicity in illustration, only one attention encoder is shown while 2 stacked attention encoders are actually set according to the fine-tuning conducted in Section V. Policy and target networks have same structure with 600-neurons hidden layers.

attention scores are continually refined leading to an identical observation received during two different interactions to have different extracted active features even if there were no major changes in the slicing infrastructure. That's what we mean by **active** features extractor.

Equation 1 is specific to the DDQN algorithm [27]. Several reasons motivate our choice of building AFEDA on top of DDQN algorithm : i) it operates in discrete action space – compared to DDPG, TD3 and Soft Actor-Critic (SAC) – allowing AFEDA to output a computing node index as an integer, ii) compared to A2C, A3C, PPO and Trust Region Policy Optimization (TRPO) which can't use a replay memory, DDQN stores past VNF placement experiences in a prioritized replay memory [28] and sample them based on their rewards temporal difference errors to train its neural networks, allowing AFEDA to be sample efficient, iii) it does not require the user to manually determine and set the bounds of the Q-values as with Rainbow, Categorical DQN and Categorical DDQN, allowing AFEDA to be instantiated without any prior knowledge or assumption on the reward function, iv) it improves on DQN algorithm by using a double neural network structure to overcome Q-values overestimation faced by DQN, allowing AFEDA to be more stable, v) it manages a good exploration/exploitation tradeoff by using a linear decay epsilon explorer to progressively switch from exploration – where most actions are chosen randomly – to exploitation involving actions based on the policy network Q-values.

C. Reward function

The environment guides AFEDA to refine its VNF placement policy by providing a reward feedback (element **(r)** in Figure 2). We set the reward function as in Equation (2) which is an extension of the one employed in [21].

$$r_t = \begin{cases} 100 \times \frac{1}{|\mathcal{L}|} \times \left(\frac{\overline{\mathcal{R}}^{cpu}_{j,i}}{\mathcal{R}^{cpu}_{i,max}} + \frac{\overline{\mathcal{R}}^{ram}_{j,i}}{\mathcal{R}^{ram}_{i,max}} + \frac{\overline{\mathcal{R}}^{stor}_{j,i}}{\mathcal{R}^{stor}_{i,max}}\right) \text{ if} \\ \text{VNF successfully placed} \\ \\ -100 \text{ otherwise} \end{cases} \tag{2}$$

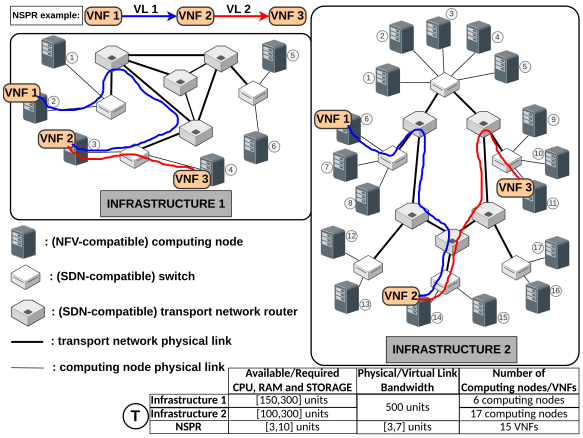
where $\mathcal{R}^{RES}_{i,max}$ is the maximum available resource RES (CPU, RAM or storage) on computing node c_i and $\mathcal{L}$ is the physical path used to match virtual link $(p_j, p_{j'})$.

V. Simulation Results

Inside the environment, the NSPRs are placed on two possible infrastructures as depicted in Figure 4. They are inspired by those introduced in previous works [29] and [19] with Infrastructure 1 having 6 computing nodes and Infrastructure 2 having 17. In Infrastructure 2, we first fine-tuned the active features extractor specific to AFEDA and proceed to an ablation study by comparing AFEDA against a conventional DDQN agent while setting all their common (hyper)parameters to be exactly the same. By manually setting environment seeds, we ensure the NSPRs (15 VNFs per NSPR) to be placed are same for all agents. The same reward function in Equation (2) was also used to guide the agents while we use LION [30] as optimizer for their neural networks. We grouped the interactions between the agents and their environments by episodes with an episode beginning when the infrastructure resources are initialized as in Table 4-T. An episode is terminated when the agent failed to place a VNF. We run each simulation 3 times and plot their results' average along with a smoothed average (for clarity) using a Savitzky-Golay filter.

A. Fine-tuning the active features extractor

1) Setting number of stacked encoders: We tested four AFEDA versions with 1, 2, 3 and 4 stacked encoders. The



	Available/Required CPU, RAM and STORAGE	Physical/Virtual Link Bandwidth	Number of Computing nodes/VNFs
Infrastructure 1	[150,300] units	500 units	6 computing nodes
Infrastructure 2	[100,300] units		17 computing nodes
NSPR	[3,10] units	[3,7] units	15 VNFs

Fig. 4: Physical infrastructures set in our simulations : Infrastructure 1 has 6 computing nodes and Infrastructure 2 has 17 computing nodes. Only one infrastructure is loaded at a time in the environment. For illustration purpose, we provide some possible deployments from an agent trying to satisfy an NSPR with 3 VNFs.

performance of each version is visible in Figure 5-a. From 1 to approximately 120th episode, all agents exhibit poor performance which is a consequence of the exploration phase during which agents act almost randomly to gather some experiences and discover the VNF placement possibilities. As soon as they reach their minimum number of 25,000 experiences, they start training their neural networks and switch progressively from a random VNF placement policy to a greedy policy characterized by a surge in performance as observed between the 120th and 350th episode with the 2-encoder agent leading, followed by the 1 and 3-encoder agents, and lastly, the 4-encoder agent. The trend continues and span approximately 350 to 900 episodes. Contrary to intuition, there isn't a linear relationship between adding more encoders and the performance of AFEDA as the version with 4 encoders is even more unstable than its counterparts (highlights by its unstable background average curve). So, we selected the 2 encoders version.

2) Setting the regularization dropout value: Four versions were tested with dropout regularization values of 0, 0.2, 0.4, and 0.6 respectively. Regarding the performance exhibited in Figure 5-b and using the observed instability in the background average curves, we hesitate between the version with dropout 0 and the one with 0.4. Given that dropout aims to help the agent to not overfit, we selected the version of AFEDA with dropout=0.4.

3) Setting the number of attention heads: To determining the number of heads to use in each encoder, we tested three versions of AFEDA all having 2 encoders in which dropout is 0.4. From Figure 6-a, the agents with 3 and 5 attention heads exhibit the highest performance with an overlapping observed between the 700th and 900th episodes. On the other hand, the agent with 1 head demonstrates significant instability. Considering the more frequent instability observed in the agent with 3 heads (blue background curve), we finally opted for the agent with 5 heads. Note that with 5 heads version, we applied single-unit zero-padding to each computing node's token to ensure its size is divisible by the number of heads (mandatory).

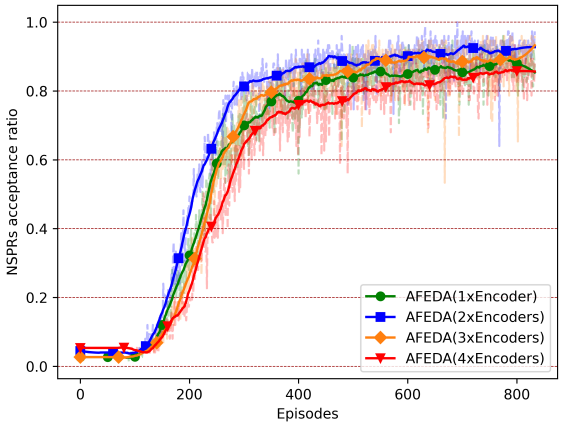
B. Comparing AFEDA against DDQN

The common practice in existing works is to flatten the received observations from environment and input them directly (or after applying a z-score normalization) into a DRL agent. Thus, we compared AFEDA against two versions of DDQN agent : a version DDQN{+n} where the observations are z-normalized and a version DDQN{-n} without any normalization.

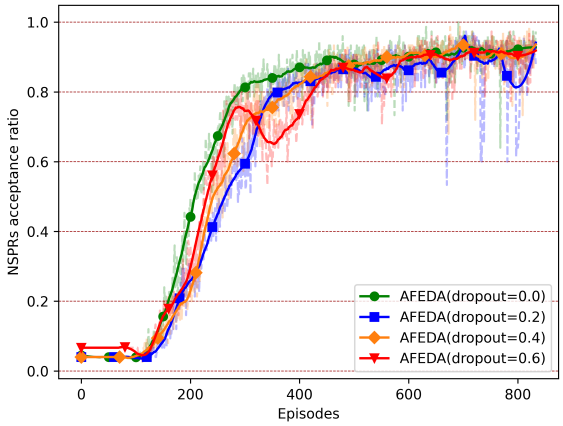
1) Acceptance ratio in Infrastructure 1: The acceptance ratio of each agent is visible in Figure 6-b whcih spans 600 episodes. From 1st episode, AFEDA's performance keeps growing until reaching stability around the 300th episode. Around the 370th episode, both DDQN agents converge towards an acceptance ratio of about 0.86, representing approximately 14% fewer successful NSPRs compared to AFEDA. Further policy refinement won't change that trend anymore until the end of the episodes. So, it means even in a small infrastructure with only 6 computing nodes, AFEDA's active features extractor enabled it to build a slightly better VNF placement policy than both DDQN versions. Through the background average curves, we can even conclude AFEDA learns a more stable policy.

2) Acceptance ratio in Infrastructure 2: By reconducting the same simulations in Infrastructure 2 which has 1.8x more computing nodes than Infrastructure 1, we came up with the performance in Figure 7-a covering 800 episodes. The difference of 11 computing nodes in Infrastructure 2 compared to Infrastructure 1 widened the performance gap between these three agents. AFEDA demonstrated the highest performance, followed by DDQN{+n} and DDQN{-n}. The performance gap is so significant that around the 400th episode, AFEDA placed 21% and 57% more NSPRs than DDQN{+n} and DDQN{-n}, respectively. Around the 600th episode, the gap widens further, with AFEDA having an acceptance ratio 31% and 59% higher than DDQN{+n} and DDQN{-n} respectively. The results allow to draw the conclusion that even in same conditions, active features help AFEDA to increase slices acceptance ratio while stabilizing its learned VNF placement policy.

3) Mean NSPR processing time: There is a tradeoff to be aware of. We compared the average time required by AFEDA and DDQN to process an NSPR. The results shown in Figure 7-b indicate that, on average, AFEDA took 15.34 and 16.76 milliseconds to process an NSPR in Infrastructures 1 and 2 respectively, while DDQN took 8.68 and 8.90 milliseconds, respectively. It means AFEDA's performance comes with a doubling of the average NSPR processing time compared to DDQN. This is partly due to the overhead of the active features extractor and partly due to the need to replicate the VNF requirements into each computing node's token, increasing the amount of data processed by AFEDA. DDQN, on the other hand, processes its observations as a single block containing

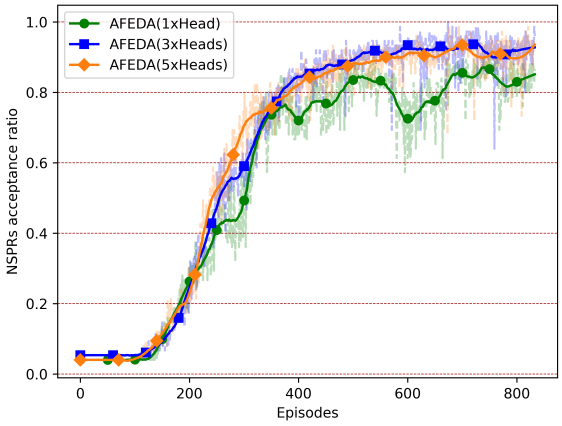


(a) Varying the number of stacked encoders.

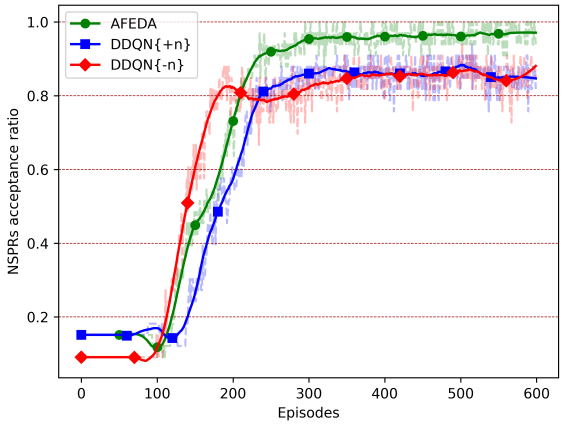


(b) Varying the regularization dropout value in each encoder.

Fig. 5: Fine-tuning AFEDA's active features extractor.



(a) Varying the number of heads in each encoder.



(b) AFEDA vs DDQN in Infrastructure 1.

Fig. 6: **Left:** Fine-tuning AFEDA's active features extractor; **Right:** Comparing NSPRs acceptance ratio.

only one copy of the VNF requirements, which results in a lower processing time.

VI. Conclusion

In this paper, we introduce AFEDA, a DDQN agent that cleverly repurposes the Transformer architecture to extract active features to learn a robust and more stable VNF placement policy. By interacting with a DRL environment containing the physical infrastructure, AFEDA was able to increase slices acceptance ratio compared fairly to two baseline versions of the DDQN agent trying to deploy the same set of network slices using the same reward function and slicing infrastructures. The active features extractor we introduced in DDQN agent to design AFEDA can be applied to other DRL algorithms. In future work, we plan to conduct and validate our simulations on bigger slicing infrastructures to test AFEDA's scalability. We also plan to move towards multi-domain network slicing scenarios by introducing a distributed version of AFEDA.

References

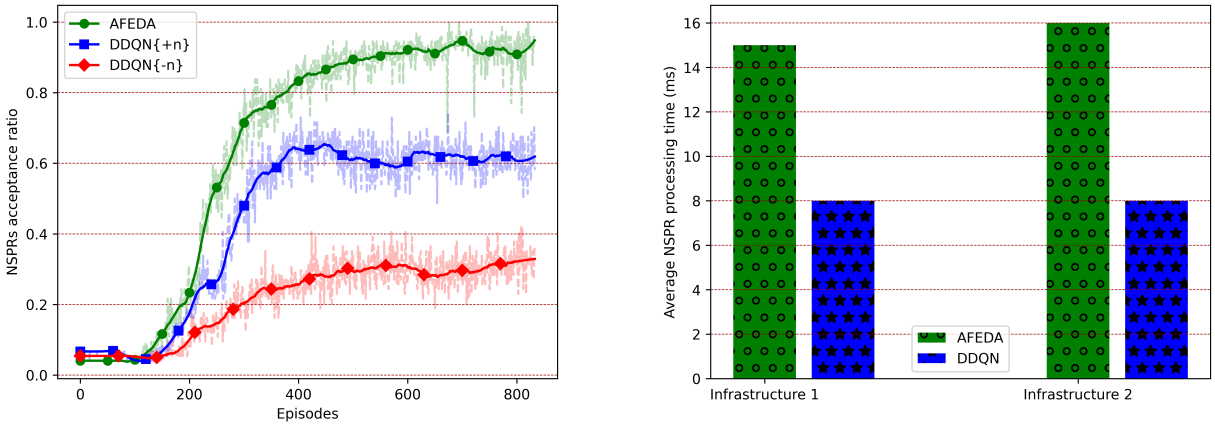
[1] S. Wijethilaka and M. Liyanage, "Survey on network slicing for internet of things realization in 5g networks," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 957–994, 2021.

[2] 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.

[3] N. Alliance, ""description of network slicing concept," ngmn 5g p, vol. 1, no. 1," 2016.

[4] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, nov 2017.



(a) AFEDA vs DDQN in Infrastructure 2.

(b) NSPR average processing time for AFEDA and DDQN.

Fig. 7: **Left:** Comparing NSPRs acceptance ratio **Right:** Time required to deploy all VNFs within an NSPR.

[6] N. Sen and A. F. A, "Intelligent admission and placement of o-ran slices using deep reinforcement learning," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pp. 307–311, 2022.

[7] A. Awad Abdellatif, A. Abo-Eleneen, A. Mohamed, A. Erbad, N. V. Navkar, and M. Guizani, "Intelligent-slicing: An ai-assisted network slicing framework for 5g-and-beyond networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1024–1039, 2023.

[8] J. Guo, G. Zhu, D. Zhang, and C. Xu, "Resource management algorithm for slicing function in 5g network slicing," in *2023 5th International Conference on Natural Language Processing (ICNLP)*, pp. 367–372, 2023.

[9] T. Dong, Z. Zhuang, Q. Qi, J. Wang, H. Sun, F. R. Yu, T. Sun, C. Zhou, and J. Liao, "Intelligent joint network slicing and routing via gcn-powered multi-task deep reinforcement learning," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 2, pp. 1269–1286, 2022.

[10] R. Huang, M. Guo, C. Gu, S. He, J. Chen, and M. Sun, "Toward scalable and efficient hierarchical deep reinforcement learning for 5g ran slicing," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 4, pp. 2153–2162, 2023.

[11] D. Xiao, S. Chen, W. Ni, J. Zhang, A. Zhang, and R. Liu, "A sub-action aided deep reinforcement learning framework for latency-sensitive network slicing," *Computer Networks*, vol. 217, p. 109279, 2022.

[12] F. Mason, G. Nencioni, and A. Zanella, "A multi-agent reinforcement learning architecture for network slicing orchestration," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, pp. 1–8, 2021.

[13] 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.

[14] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "Deepvine: Virtual network embedding with deep reinforcement learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 879–885, 2019.

[15] A. Bouroudi, A. Outtagarts, and Y. Hadjadj-Aoul, "Dynamic machine learning algorithm selection for network slicing in beyond 5g networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pp. 314–316, 2023.

[16] A. Bouroudi, A. Outtagarts, and Y. Hadjadj-Aoul, "Robust deep reinforcement learning algorithm for vnf-fg embedding," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pp. 351–354, 2022.

[17] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, 2019.

[18] H. Yao, B. Zhang, P. Zhang, S. Wu, C. Jiang, and S. Guo, "Rdam: A reinforcement learning based dynamic attribute matrix representation for virtual network embedding," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 901–914, 2021.

[19] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement," 2021.

[20] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Controlled deep reinforcement learning for optimized slice placement," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pp. 20–22, 2021.

[21] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "DRL-based Slice Placement Under Non-Stationary Conditions," in *CNSM 2021 - 17th International Conference on Network and Service Management*, (Izmir, Turkey), Oct. 2021.

[22] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Drl-based slice placement under realistic network load conditions," 2021.

[23] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[24] J. Gil Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[25] H. Bai, Y. Zhang, Z. Zhang, and S. Yuan, "Latency equalization policy of end-to-end network slicing based on reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 88–103, 2023.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[27] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[29] J. J. A. Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Location-based data model for optimized network slice placement," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pp. 404–412, 2020.

[30] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le, "Symbolic discovery of optimization algorithms," 2023.