computers

MDPI

Article

Reinforcement-Learning-Based Edge Offloading Orchestration in Computing Continuum

Ioana Ramona Martin [1], Gabriel Ioan Arcas [2] and Tudor Cioara [1,*]

[1] Computer Science Department, Technical University of Cluj-Napoca, Memorandumului 28, 400114 Cluj-Napoca, Romania; martin.io.ioana@student.utcluj.ro
[2] Bosch Engineering Center, 400158 Cluj-Napoca, Romania; gabriel.arcas@ro.bosch.com
* Correspondence: tudor.cioara@cs.utcluj.ro

Abstract: The AI-driven applications and large data generated by IoT devices connected to large-scale utility infrastructures pose significant operational challenges, including increased latency, communication overhead, and computational imbalances. Addressing these is essential to shift the workloads from the cloud to the edge and across the entire computing continuum. However, to achieve this, significant challenges must still be addressed, particularly in decision making to manage the trade-offs associated with workload offloading. In this paper, we propose a task-offloading solution using Reinforcement Learning (RL) to dynamically balance workloads and reduce overloads. We have chosen the Deep Q-Learning algorithm and adapted it to our workload offloading problem. The reward system considers the node's computational state and type to increase the utilization of the computational resources while minimizing latency and bandwidth utilization. A knowledge graph model of the computing continuum infrastructure is used to address environment modeling challenges and facilitate RL. The learning agent's performance was evaluated using different hyperparameter configurations and varying episode lengths or knowledge graph model sizes. Results show that for a better learning experience, a low, steady learning rate and a large buffer size are important. Additionally, it offers strong convergence features, with relevant workload tasks and node pairs identified after each learning episode. It also demonstrates good scalability, as the number of offloading pairs and actions increases with the size of the knowledge graph and the episode count.

Keywords: reinforcement learning; task offloading; Deep Q-Learning; knowledge graphs; computing continuum

Citation: Martin, I.R.; Arcas, G.I.; Cioara, T. Reinforcement-Learning-Based Edge Offloading Orchestration in Computing Continuum. *Computers* **2024**, *13*, 295. https://doi.org/10.3390/computers13110295

Academic Editor: Riduan Abid

Received: 2 October 2024
Revised: 1 November 2024
Accepted: 11 November 2024
Published: 14 November 2024

1. Introduction

The large volume of data generated by IoT devices connected to large-scale utility infrastructures such as the smart grid poses significant operational challenges, including increased latency, communication overhead, and computational imbalances, which negatively affect the overall performance and reliability of the management process [1]. To address these issues, it is essential to shift management workloads from cloud-based infrastructure to the edge and across the entire computing continuum [2]. The computing continuum divides the workloads according to their computational characteristics over three layers: edge, fog, and cloud, each of them having different capabilities [3,4]. The edge layer is closer to the physical infrastructure providing low latency and fast response time, the fog layer brings the cloud closer to the edge while maintaining a relatively low latency and the cloud layer has the highest computational power, being able to sustain complex tasks. Workload task offloading and orchestration aim to move the workload across the different layers to increase processing efficiency, and decision making in case of large-scale infrastructure management. With the advent of AI-driven applications, their data gathering and ingestion requirements are becoming increasingly complex [5]. If a large amount of

data needs to be moved to the cloud for AI processing, it can generate increased latency which may not be feasible for specific applications like the management of large-scale infrastructures. In this context, the computing continuum can enable the usage of edge–fog computing and communication models to ensure the integration of resources and optimal latency and bandwidth management [6].

However, to achieve this, significant challenges must still be addressed, particularly in decision making to effectively manage the trade-offs associated with workload offloading [7]. Edge devices often have limited computing, storage, and power resources, making it difficult to handle heavy workloads efficiently. Effectively allocating resources based on changing workloads and demand is complex and requires sophisticated orchestration decision making. Several decision-making techniques can be employed to determine when and what to offload [8]. The choice of technique depends on factors such as the workload characteristics, the dynamic nature of workloads, and the specific goals of offloading. State-of-the-art solutions address optimal decision making for workload offloading by leveraging heuristic algorithms [9,10]. They are efficient in addressing the complexity of the decision space and provide good enough solutions within a reasonable time frame. However, it is challenging to define the initial model and constraints to be addressed by heuristics, and afterward, to continuously update the model considering the updated state of cloud–fog–edge infrastructure, as provided by the monitoring solutions [11].

Lately, other techniques, such as machine learning algorithms, have been explored for workload offloading in various contexts [12]. In particular, RL algorithms are promising solutions for task offloading in graph-based infrastructures, as they can adapt to dynamic environments and handle diverse data sources effectively [13]. They can learn from previous experiences or existing data, allowing for more adaptive decision making considering factors such as the available computational resources and communication network conditions [14]. However, RL can bring high decision-making overhead when dealing with large search spaces and numerous actions to simulate [15]. In addition, the computing continuum provides the possibility of automating infrastructure deployments. The deployments can be performed on various infrastructures, ranging from edge to cloud, increasing the efficiency and lowering the complexity [4]. Moreover, the workloads spread over the computing continuum can be containerized and adapted, taking into account the layer where they might belong (edge, fog, or cloud), their characteristics, and restrictions [16].

Nevertheless, there are other techniques, not related to RL, which are utilized in task offloading problems. For example, there are offloading algorithms based on game theory, which aim to achieve offloading efficiently, while also preventing interference [17]. Moreover, there is an offloading algorithm based on SDN (Software Defined Networking), which takes into account latency and throughput to perform optimal task offloading [18]. In addition, for maximizing the number of offloading tasks, an approach would consist of heuristics algorithms based on hypergraphs [19]. Another algorithm that implies high complexity is a heuristic one based on PSO (particle swarm optimization), which tries to decrease the probability of service failure [20]. Furthermore, offloading can be performed using Genetic Algorithms (GA) and Greedy strategies, aiming to reduce the communication cost [21].

In this paper, we propose a task-offloading solution leveraging Reinforcement Learning (RL), which will dynamically adapt the system configuration to balance workloads and reduce overloads. Through RL, the system continuously learns from the environment, improving efficiency by offloading tasks to less overloaded nodes, ensuring optimal performance across the computing continuum. We have chosen Deep Q-Learning as the RL algorithm and we have specially adapted it to the requirements of our task offloading problem. The reward was modeled considering not only the computational state of the nodes (i.e., overloaded or underloaded) based on their computational and network characteristics but also their type and sub-type. To address challenges related to environment modeling and to facilitate RL learning by action simulations we have leveraged a knowledge graph (KG) model of the computing continuum infrastructure [22]. The KG offers good representation features for computational entities and the relationships established among them, as

well as their properties, enabling the agent to reason on the infrastructure state. During the learning process, the KG (environment) will be updated to reflect the edge–fog–cloud infrastructure state, leading to a more efficient configuration than in the beginning with less overloaded nodes, if not none. Moreover, the KG infrastructure is continuously updated during the RL process, with each offloading action performed, to increase the workload balance among the computing continuum nodes.

The rest of the paper is structured in the following manner. Section 2 presents the related work on RL applications for workload offloading in the computing continuum. Section 3 describes our proposed task and orchestration solution. Section 4 presents the evaluation results in a use case. Section 5 concludes the paper and presents future work.

2. Related Work

The computing continuum and distribution of workload and computational resources from the cloud toward the edge play important roles in addressing the limitations of traditional centralized management in the context of big data and AI-driven applications. Kiss et al. [23] proposed a system of interdependent edge swarms managed by decentralized Orchestration Agents, using distributed intelligence to match the availability of the resources. In addition, blockchain is utilized to ensure security and traceability, while a digital twin aims to improve the behavior through predictive feedback. Task offloading deals with moving demanding computational tasks from overloaded edge devices to a more powerful network entity, to improve the energy management in systems from smart cities [24]. Since large-scale utilities such as smart grids require low latency for the majority of their operations and an extremely good resource allocation, task offloading is a key technique to be utilized in this case, by leveraging mobile edge computing technologies. For this particular use case, a deep Q-network (DQN) was used in favor of obtaining better task offloading decisions, lower latency, and, in consequence, higher efficiency in smart grid environments [25]. Moreover, Q-learning algorithms can be applied to Edge-Based Heterogeneous Task Specific Offloading (HTSO) with Genetic Algorithm (GA) optimization to decrease latency, response time, and energy usage levels for smart grids. This plays a crucial role in smart city infrastructures, where it can increase performance considerably [26]. For cloud-based systems, low latency and high-reliability conditions cannot be achieved in specific cases. For these, the limitation above can be surpassed by merging the optimization of task offloading with service caching in edge-based smart grids via a Collaborative Computing Offloading and Resource Allocation Method (CCORAM) [27]. Furthermore, algorithms such as the Dynamic Switching Algorithm are utilized to strategically offload a certain task in the cloud or edge, taking into account the real-time conditions of a system. This ensures a balance between the low latency and bandwidth that is needed and the huge volume of data produced by the smart IoT applications [28]. In the case of fog, the middle layer of the computing continuum, bridging the cloud and the edge, algorithms based on reinforcement learning are used to distribute the computational load between the nodes. This distribution of load is highly beneficial for fog computing since it optimizes the traffic and supports applications that require real-time response [29]. For Multi-access Edge Computing, the increasing number of devices may become a serious issue because the decision-making process for offloading cannot be based only on data and infrastructure. For this, reverse offloading solutions are preferred since they consider both the task completion time and the energy consumption, lowering the overall usage of the system [30]. On top of that, Gated Recurrent Unit (GRU) or dueling and double DQN techniques are also used in Multi-access Edge Computing for offloading tasks between the edge nodes, with the scope of reducing costs. These imply that the task offloading decisions are made independently across the system [31]. Furthermore, edge–fog–cloud models and computational orchestration are techniques able to improve task offloading in smart grids, by leveraging characteristics such as network and computational requirements for offloading, metaheuristics, Artificial Intelligence, or Reinforcement Learning [8].

For optimizing application performance, a solution based on the construction of dynamic KG modeling computational resources is proposed. This is crucial in day-ahead scheduling decisions, leveraging KG-specific techniques such as relation extraction and knowledge fusion or dynamic updates [32]. In addition, an energy knowledge graph (EKG) is proposed for smart energy services, given the fact that it can provide a schema capable of reusing knowledge resources related to energy and it can meet the requirements imposed by the decentralized and large-scale nature of the grid [33]. Likewise, the Whale Optimization Algorithm (WOA) can be utilized in a directed acyclic graph environment, modeling a smart grid service infrastructure, to produce cloud–edge offloading decisions, while decreasing the average execution time [9]. Multi-task Offloading via graph representation is another use case of the graphs in task offloading. This representation takes into account the task features, along with the network conditions and the resources available for offloading, ensuring optimization of the resource usage in heterogeneous networks [34].

RL algorithms can be powerful assets in task offloading for graph-based infrastructures since they can adapt to dynamic environments or they can handle gracefully data from heterogeneous sources. Some of the approaches found in the literature include a mechanism based on graph neural networks. In this case, the environment is modeled as an acyclic graph providing lower latency compared to other similar approaches [13]. Deep reinforcement learning frameworks are used for increasing the efficiency of both task allocations and resource management in Multi-Access Edge Computing [35]. In addition, a curriculum attention-weighted graph recurrent network-based multi-agent actor-critic model is proposed to overcome limitations, such as resource constraints in cases where a large number of edge devices are offloading tasks to an edge server. This method proved to increase task completion and significantly reduce costs [36]. Moreover, graph reinforcement learning-based early-exit mechanisms are used to balance communication, computational load, and inference accuracy of workload offloading. The solution based on early exit mechanisms turned out to be more accurate than other traditional methods and to be better at decision making [37]. Furthermore, graph convolutional network-based reinforcement learning methods represent a good strategy in the area of task offloading, since they can defeat the limitations related to the incomplete Markov models. This is achieved by extracting the features from the tasks, modeled as a directed acyclic graph, with the help of a Graph Convolutional Network, and then constructing a complete Markov model for offloading, with the scope of delivering continuous and efficient offloading decisions [38]. To address the overloading of the edge servers and the high response latency, Xu et al. propose TransEdge, a task offloading system, using graph networks and Deep Reinforcement Learning (DRL) in edge computing-enabled transportation systems. To be more precise, this solution uses an adaptive node placement algorithm to minimize transmission latency, the DRL to enhance the offloading accuracy, and a greedy task forwarding strategy to overcome the system instability [39]. The offloading process can be quite challenging for dependent tasks in the dynamic mobile edge. This can be addressed using DRL, which models task graph scheduling as a Markov Decision Process to adapt to the varying computation capabilities of edge devices, improving user experience [40].

3. Deep Q-Learning Edge Orchestration

We propose a Deep Q-Learning edge orchestration that leverages an agent that learns the optimal solution by trying every possible combination of actions in a what-if manner and evaluating the associated rewards and penalties (see Figure 1). To represent the computing continuum environment in which the agent operates we have used a knowledge graph-based model that describes the resources on the edge, fog, and cloud, and the IoT assets layers. We have leveraged the computing continuum model described in one of our previous publications [22] in this scope. A knowledge graph offers distinct advantages over simpler models like tables or basic graphs, particularly for managing the complex and dynamic needs of edge–fog–cloud environments. Unlike tables, which only display basic resource details, or Directed Acyclic Graphs (DAGs), which capture connections

without depth, a KG incorporates semantic relationships essential for improved offloading decisions. In previous work using DAGs [9], we found that they effectively modeled structural dependencies but struggled to capture complex interdependencies and context-sensitive information. This limitation restricted the system's ability to make adaptive, real-time decisions across distributed resources. In contrast, KGs leverage reasoning capabilities that enable context-aware decisions, crucial in environments where resource availability and network conditions are constantly changing. This semantic feature allows KGs to improve response times and resource efficiency by representing not just "what" resources are connected, but also "how" and "why" they interact, such as resource affinity or priority during high-demand periods. While managing a KG involves additional setup and operational costs, it is a worthwhile investment, as simpler structures like DAGs often lack the adaptability and insight needed for optimal performance in interconnected systems. Therefore, the precision, reasoning capability, and adaptability of KGs make them essential for enhancing the efficiency and responsiveness of edge–fog–cloud networks. In this scope, our solution combines deep neural networks with KGs, providing a large state and action space and the ability to deal with many nodes and dynamic edge environments.

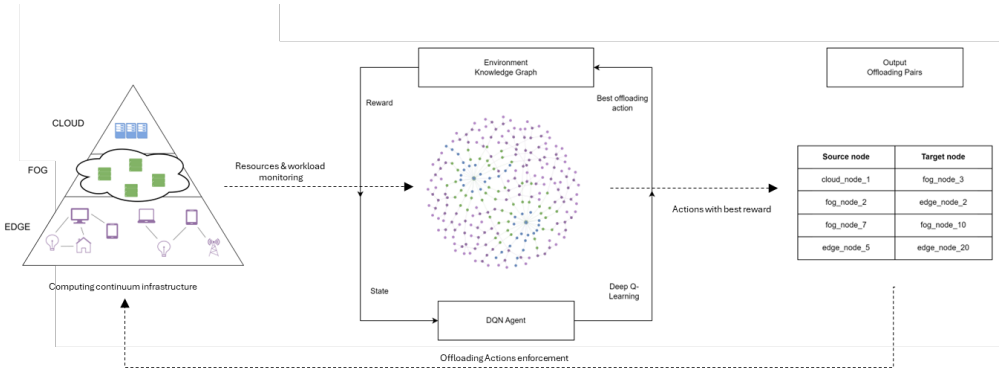


Figure 1. RL-based offloading solution.

The agent is the component that interacts directly with the environment with actions and rewards. Its goal is to learn, gain rewards, not penalties, and maximize the reward. In our case, the agent is based on Deep Q-learning using neural networks to estimate the expected future reward for any action. It has a Q-Network for the Q-values and a Target Network for the target Q-values. The Deep Q-learning Algorithm 1 starts with initializing the experience replay buffer and the neural networks (lines 1–2). The experience replay buffer stores data about the agent experiences, while the two neural networks are used in the learning process. The Q-network computes the expected rewards based on the agent's observations on the KG-based environment and the target network is a copy of the first one, updated at a certain number of steps. The hyperparameters of the Q-network are also set in the initial phase. Afterward, at every step, a random edge offloading action is selected and executed on the KG-based environment. A tuple of the offloading action, the reward, and the observations is stored in the experience replay buffer. If the current step is a learning (training) step for the agent, a random batch of training data are selected from the experience replay buffer. For each piece of information from the chosen batch, the future reward has to be calculated (see lines 12–14). The gradient descent is performed after the reward computation, to minimize the loss. In the end, it is checked whether the target network has to be updated and updated appropriately. Therefore, the DQN agent learns to choose the actions that will give it the positive rewards. It will change the state of the environment for positive actions when it will also perform task offloading between the nodes from the KG. For all the pairs, the agent swaps are the actual result for the offloading problem.

Algorithm 1 Deep Q-Learning Edge Orchestration

Initialize experience replay buffer D, Q-network Q and target Q-network q.
Set q's parameters to Q.
for each step **do**
    With probability ϵ, select a random action.
    Otherwise, select the action with the highest expected future reward according to $Q(s, a)$.
    Execute action a, collect reward r and observation $t + 1$ from the environment.
    Store the transition $obs(t)$, $action$, $reward$, $obs(t + 1)$ in D.
    if current step is a training step **then**
        Collect a random batch of transitions from D.
        for each transition j **do**
            Set future reward $y(j) = \begin{cases} r(j) & \text{if episode ended at } j \\ r(j) + \gamma * q(s', a') & \text{otherwise} \end{cases}$
        end for
        Perform gradient descent with respect to the loss function to update Q.
        if current step mod C == 0 **then**
            Set q's parameters to Q.
        end if
    end if
end for

We adapted the Deep Q-Learning algorithm, for our edge offloading and orchestration case, in terms of how the reward function is crafted to be in line with the offloading conditions and the custom environment. The environment is represented by a unique KG model of the computing continuum, with its nodes and edges, also including all of their characteristics (1). Therefore, we formalize the environment as:

$$KG = (N, W) \tag{1}$$

where N is the set of computational continuum nodes and W is the set of workload tasks executed by the nodes. Each node $n_i \in N$ is characterized by certain features such as the available CPU, RAM, data volume, latency, and bandwidth. The computational environment changes based on the workload task offloading performed by the agent. In this way, the agent is enabled to interact permanently with the computational environment receiving rewards or penalties for updating it by moving workload tasks from node to node.

The action space and the observation space were defined for the considered environment. The action space consists of all the actions that the agent can take. In our case, it is all the pairs of nodes that can be chosen from the graph to be candidates for task offloading, where each node represents an aggregate of multiple tasks, perceived as a whole workload. At the same time, the agent's observation space is the edge nodes' characteristics, since they play an important role in the offloading decision and are used in computing the reward. Therefore, the agent takes a pair of nodes (i.e., source node and target node) and checks if all of the required conditions for offloading are met. In the affirmative case, the agent will move the entire load from one node to another, updating the KG environment. This is a positive action that will be rewarded, but for any other action that does not lead to a successful offloading, a penalty is enforced. Formally, the action space A is defined as the set of all possible offloading scenarios, consisting of a pair of nodes (n_i, n_j), where $n_i \in N$ is a source node, $n_j \in N$ is a target node, and t_k is a task with associated computational requirements. The action space is defined as:

$$A = \{a_t = (n_i, n_j, t_k) \mid n_i, n_j \in N \text{ and } n_i \neq n_j\} \tag{2}$$

and consists of all the offloading actions $a_t \in A$ of tasks from the source node towards the target node.

The state $s_t \in S$ is a computing continuum environment snapshot at a given step during the learning process. Therefore, a state is represented by the nodes and their available computational and data network resources for executing the workload:

$$s_t = \{\text{CPU}(n_i), \text{RAM}(n_i), \text{latency}(n_i), \dots \text{ for each node } n_i \in N\} \quad (3)$$

The offloading actions executed in a what-if manner by the agent are changing the state of the computing continuum environment. This is represented by the change from the current state s_t to the next state s_{t+1}, when an action (offloading action) a_t is applied. Formally, this is expressed by the following equation:

$$a_t(s_t) -> s_{t+1} \quad (4)$$

The reward function is defined on top of the environment to ensure that the agent will learn from its actions, by receiving a reward or a penalty. Therefore, for workload task offloading, the agent will receive a reward only if it successfully finds a pair of source and destination nodes that meet the offloading conditions and updates the KG accordingly. In different circumstances, it will receive a penalty, so that it knows it was not a good action for the learning process. In the evaluation phase, we have used:

$$r = \begin{cases} 0 & \text{the initial agent's reward} \\ 1 & \text{if the offloading conditions are met} \\ -1 & \text{if the offloading conditions are not met} \end{cases} \quad (5)$$

During the learning process, the agent will pick two random nodes, a source node, and a target node as workload-offloading candidates. The offloading conditions are evaluated. The agent will receive a positive reward if the source node and the destination node are not the same:

$$n_i \neq n_j \quad (6)$$

and the nodes have the expected types and sub-types, in line with the computing continuum layers (see Table 1). The aforementioned table brings more clarity to the fact that the offloading actions can be performed within the same layer of computing continuum (e.g., edge to edge) or within different layers (e.g., cloud to fog and fog to edge). This feature is particularly important since the solution aims to move the extensive computational load from the cloud to the edges of the network.

Table 1. Offloading node types/sub-types.

Source Node	Target Node
Type: cloud	Type: fog
Type: fog	Type: edge, sub-type: not IoT asset
Type: edge, sub-type: IoT asset	Type: edge, sub-type: edge device
Type: cloud	Type: cloud
Type: fog	Type: fog
Type: edge, sub-type: not IoT asset	Type: edge, sub-type: not IoT asset
Sub-type: IoT asset	Sub-type: IoT asset

Each node from the custom KG has a certain set of properties, both computational (e.g., CPU, RAM, etc.) and network-related (e.g., latency, bandwidth, or response time). The values of these characteristics play a key role in the offloading decisions, dictating if a node is overloaded, underloaded, and capable of receiving a certain amount of extra

workload. In particular, the requirements for the source node n_i for workload offloading to be overloaded are:

$$\text{latency}(n_i) > l_T, \text{bandwidth}(n_i) > b_T, \text{response time}(n_i) > r_T \quad (7)$$

The above-mentioned thresholds are different and personalized for every type and sub-type of nodes. For instance, an edge node will be considered overloaded at a lower latency than the one of a node from the cloud layer.

Moreover, the target node n_j must meet specific computational properties to sustain the offload from the source node:

$$\begin{aligned} \text{availableCPU}(n_j) &> W_{CPU} \\ \text{availableRAM}(n_i) &> W_{RAM} \\ \text{availableDataVolume}(n_j) &> W_{DataVolume} \\ \text{availableMemory}(n_i) &> W_{Memory} \end{aligned} \quad (8)$$

4. Results

In Python, we have implemented the solution using the *Stable-Baselines3* (https://stable-baselines3.readthedocs.io/en/master/, accessed on 10 September 2024) library. The Deep Q-Learning algorithm was combined with a custom computing continuum environment implementation. The evaluation of the DQN agent (https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html, accessed on 10 September 2024) was monitored with the help of *TensorBoard* plug-in graphs (https://www.tensorflow.org/tensorboard, accessed on 10 September 2024), provided by the library.

Multiple configurations of hyperparameter values were tried to find the best one for our case. We have considered several hyperparameters that can influence the behavior of the agent during learning, the stability, or the state of the neural network. A high learning rate leads to an unstable, yet fast learning process, while a low learning rate provides more stability, but a slower process. A larger size of the experience replay buffer increases the variability of the data for the learning process. The size of the data batch extracted from the experience replay buffer influences the stability but larger sizes require more computational power. The discount factor (i.e., the Gamma parameter) tells the agent how much to value the future and immediate rewards. It ranges from 0 to 1 and if it is closer to 1, the future reward is just as valuable as the immediate one. The Tau parameter ranges from 0 to 1 and shows how the target network is updated during the learning. If it is closer to 0, it will produce soft updates, and otherwise, hard updates. The target update interval affects stability, as it defines the number of steps, after which the target network is updated with data from the Q-network. The exploration fraction dictates the balance between exploration and exploitation performed by the DQN agent in the environment. We evaluated their impact on the mean episodic reward calculated during the learning, *rollout/ep_rew_mean*, reported by *TensorBoard*.

There were eight different configurations which were firstly tested and they will be thoroughly presented in Table 2. For all of them, the number of steps per episode was 250 and the total number of learning steps was 100,000.

In Figure 2, there are graphs representing the mean reward for the configurations compared. The best configuration considering the mean reward obtained during episodes is the *Configuration 8*. Also, it provided stability to the agent, a large buffer size, so that the agent had various data for learning, a tau that produces moderated updates, a high gamma, for adding more value to the future rewards, and an exploration fraction that makes the agent shift from exploration to exploitation quickly.

Configuration 1 of the hyperparameters is very similar to *Configuration 8*, an increased learning rate being the only difference (Learning rate = 0.0001). In Figure 2a, it can be seen that the higher learning rate from the *Configuration 1* (purple) makes the training more unstable because the mean reward increases abruptly and then decreases, while for the *Con-*

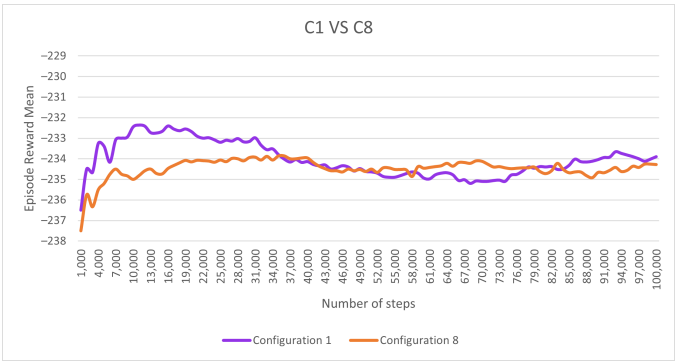
figuration 8 (orange), it increases and then it converges, which is what it is supposed to do in case the of reinforcement learning. In the case of *Configuration 2*, the exploration fraction is changed to 0.5. Therefore, the learning will be divided equally between exploration and exploitation of the environment. In Figure 2b, *Configuration 2* (light green) keeps the mean reward in the interval [−235, −233], while increasing and decreasing over the 100,000 steps of learning, which is not the desired behavior. For *Configuration 3* we minimized the buffer size providing less various training data. In Figure 2c, it can be easily visualized that *Configuration 3* (blue) is barely increasing, or converging, which provides insights that the agent is not performing properly. *Configuration 4* features a decreased batch size (Batch size = 64), and a decreased gamma (Gamma = 0.8), which still prioritizes the immediate rewards more. In Figure 2d, *Configuration 8* (orange) and *Configuration 4* (dark blue) are similar in terms of convergence. However, *Configuration 8* has a higher mean reward. *Configuration 5* has a higher learning rate and tau than *Configuration 8*, leading to harder updates of the target network. Figure 2e shows the visualization of the mean episodic reward for *Configuration 8* (orange) and *Configuration 5* (light blue). In the graph, it can be seen that the light blue line is not increasing and converging as smoothly, due to the increased learning rate, making it a less suitable configuration. In *Configuration 6*, the hyperparameter gamma was reduced to 0.5, and the exploration rate was increased to 0.2. From Figure 2f, it can be seen that *Configuration 6* (pink) is unsuitable as the learning rate is decreasing and then becoming stable, instead of increasing. Finally, *Configuration 7* is similar to *Configuration 8*, but with halved target update interval that causes instability to the learning process. In Figure 2g, *Configuration 8* (orange) and *Configuration 7* (red) are quite similar, increasing and then converging, but overall, the performance of *Configuration 8* is better, because it has a higher mean episodic reward.

Table 2. Learning hyperparameter configurations.

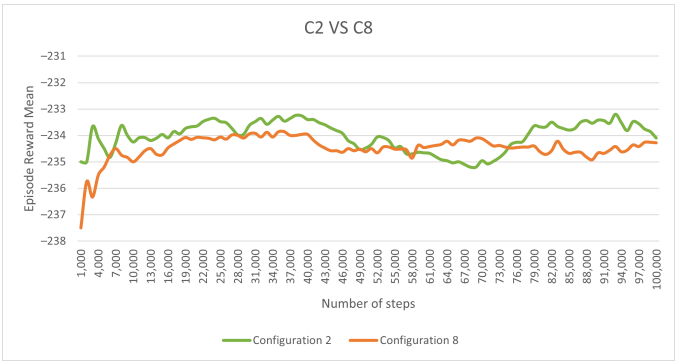
Hyperparameter	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**
Learning rate	0.0001	0.00005	0.00005	0.00005	0.0001	0.00005	0.00005	0.00005
Buffer size	200,000	200,000	10,000	200,000	200,000	200,000	200,000	200,000
Batch size	128	128	128	64	128	128	128	128
Gamma	0.99	0.99	0.99	0.8	0.99	0.5	0.99	0.99
Target update interval	10,000	10,000	10,000	10,000	10,000	10,000	5000	10,000
Exploration fraction	0.1	0.5	0.1	0.1	0.1	0.2	0.1	0.1
Tau	0.5	0.5	0.5	0.5	0.9	0.5	0.5	0.5

Additionally, we have evaluated the impact of the reward and penalty values on the agent learning process. Previously configurations were based on a +1 value for reward and a −1 value for penalty. Even though the aforementioned values are normalized and they belong to a symmetric interval, our goal was to determine the impact of other reward penalty values such as +1 for reward and 0 (zero) penalty. In case of a neutral penalty (see Figure 3), the mean reward was decreased since the agent did not receive a clear penalty.

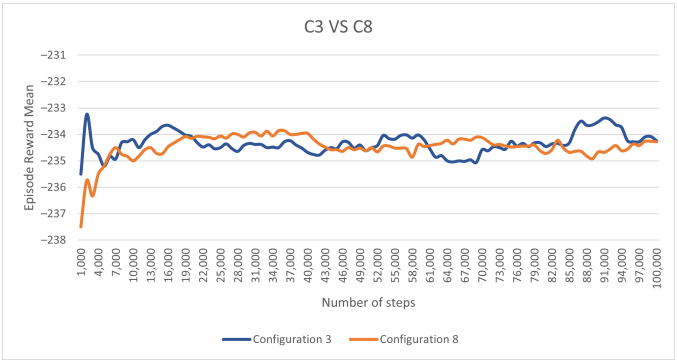
Moreover, using the best configuration of hyperparameters (i.e., *Configuration 8*), more experiments were conducted, by varying the number of steps per episode (100, 250, and 500) and the size of the knowledge graph (see Table 3). We aim to test whether an expanded KG—representing more entities and relationships—could enhance the model's understanding of task distribution and node selection. Therefore, the number of nodes of each type and sub-type was chosen so that the ratio edge–fog–cloud from real-world computing continuum environments was respected.



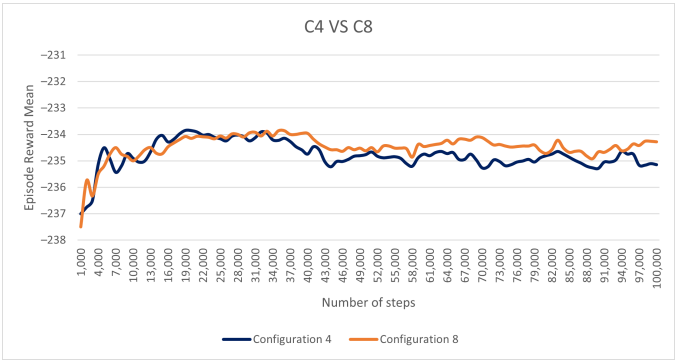
(**a**) C1 VS C8



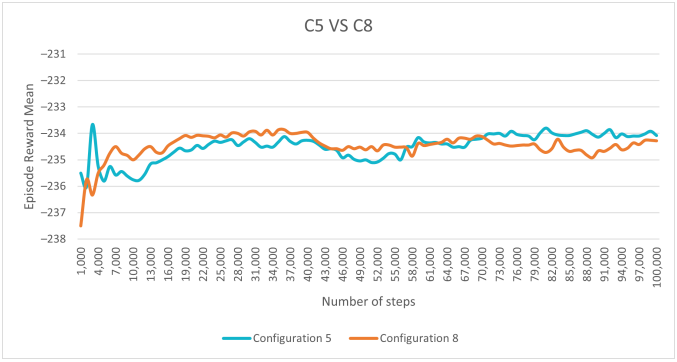
(**b**) C2 VS C8



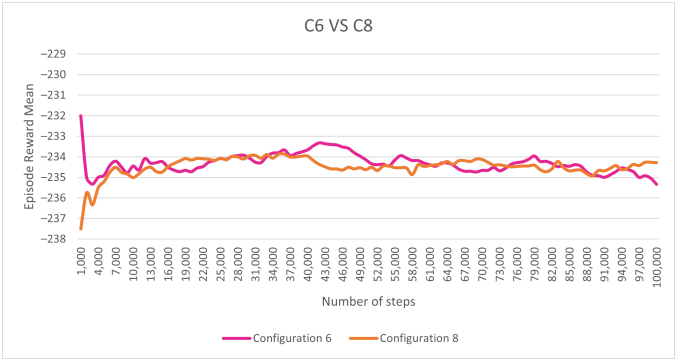
(**c**) C3 VS C8



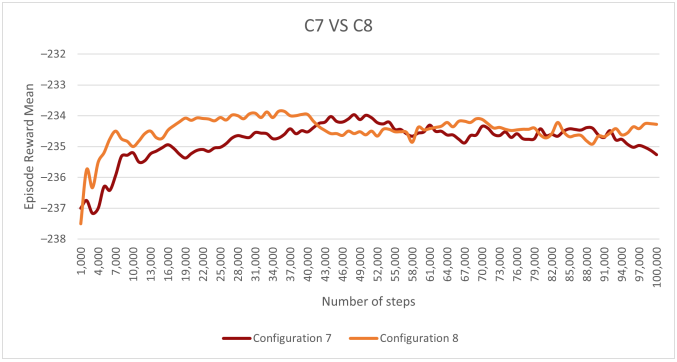
(**d**) C4 VS C8



(**e**) C5 VS C8



(**f**) C6 VS C8



(**g**) C7 VS C8

Figure 2. Hyperparameter configurations.

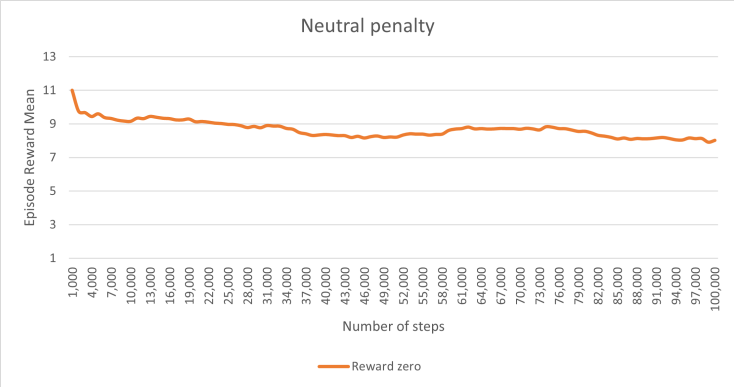


Figure 3. The mean reward evolution for a neutral penalty (0).

Table 3. Knowledge graph sizes.

Knowledge Graph	Number of Nodes
Small	120
Medium	258
Large	770

Figure 4, shows that the performance of the DQN agent for 100 steps/episode is not appropriate as the mean reward during the episode is not increasing and converging. This is true in all the small computing continuum knowledge graph (i.e., for the small graph, the yellow line, for the medium graph, the teal line, and for the large graph, the purple line).

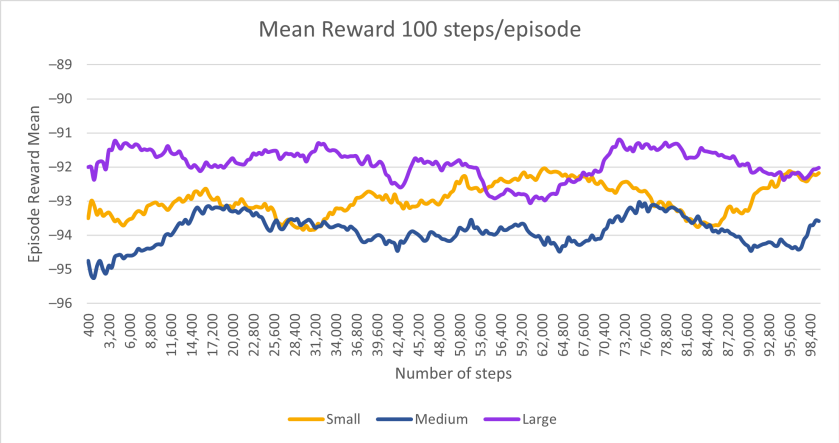


Figure 4. Mean reward for the 100 steps/episode case.

Additionally, Figure 5, shows that the loss is decreasing for all the sizes, but it is, also, directly proportional to the size of the knowledge graph.

The learning evaluation for 250 steps/episode (see Figure 6) reveals that this choice is better than the previous one. Only for the small graph (pink line), the line is firstly decreasing and then getting back on track, but for the medium graph (orange) and large graph (yellow), it is behaving as expected. Also, from the graph, it can be seen that the reward is the highest for the larger knowledge graph since it has more nodes/offloading options for the agent to choose from.

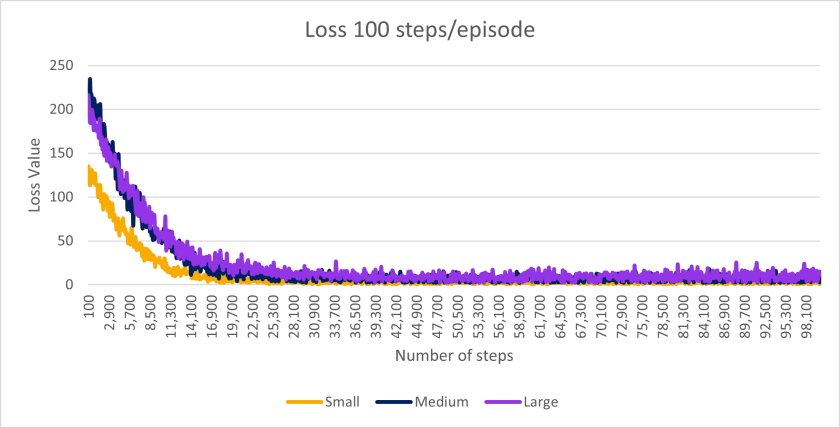


Figure 5. Loss for the 100 steps/episode case.

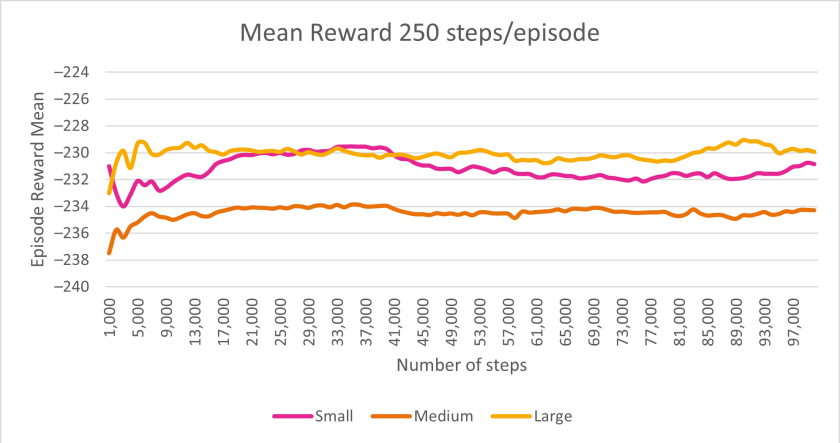


Figure 6. The mean reward for the 250 steps/episode case.

The previous trend of the loss increasing with the size of the knowledge graph is maintained (see Figure 7). However, the overall final loss is slightly higher.

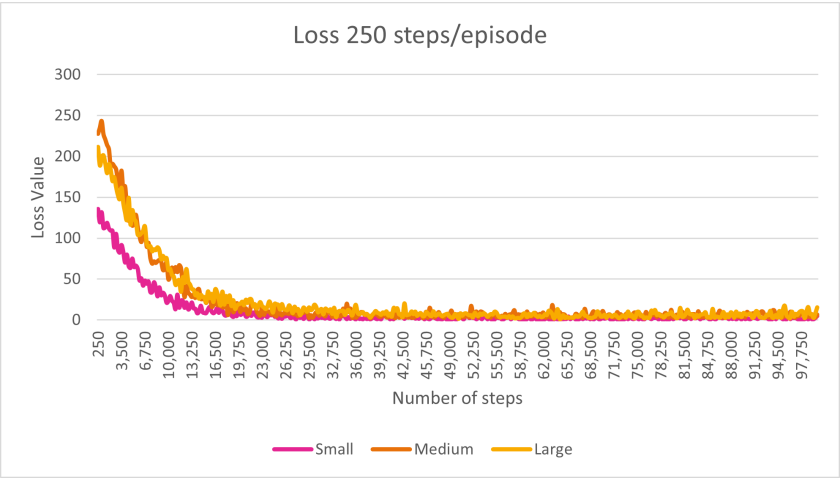


Figure 7. Loss for the 250 steps/episode case.

The best performance for all the knowledge graph sizes was for the 500 steps/episode (see Figure 8), since, for all, there is an increase, and then the lines become stable. Just like in the previous case, the large graph (pink) has a higher mean reward than the small (blue) and medium (green) ones.

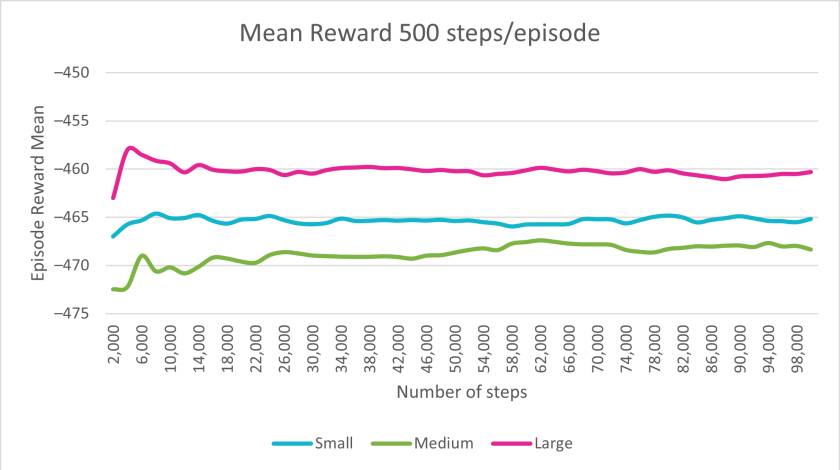


Figure 8. Mean reward for the 500 steps/episode case.

Figure 9 shows that the loss is still decreasing with size. Nevertheless, the final value for loss is higher for the small configuration, compared to the 250 steps/episode one. In addition, the loss is lower for the medium and large ones.

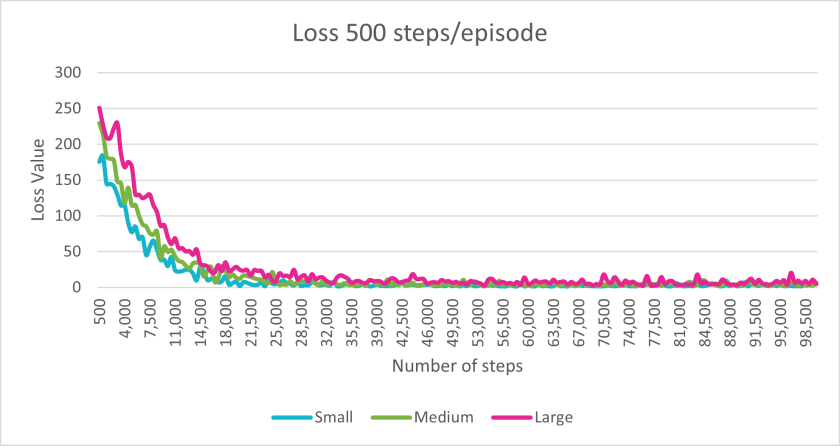


Figure 9. Loss for the 500 steps/episode case.

Table 4 provides an overview of the offloading results for the small, medium, and large knowledge graph, each trained with 100, 250, or 500 steps/episode. The results show that the number of pairs of nodes for offloading is increasing if the number of steps per episode is increasing. Moreover, if the knowledge graph has more nodes, the number of possible pairs is increasing as well, since there is a larger search space. An exception is the medium graph trained with 500 steps/episode, where the number of pairs is lower than the one for the small knowledge graph. Most of the offloaded pairs were made of nodes within the same layer of the computing continuum. However, during the learning process, the agent found cross-layer pairs for offloading as well.

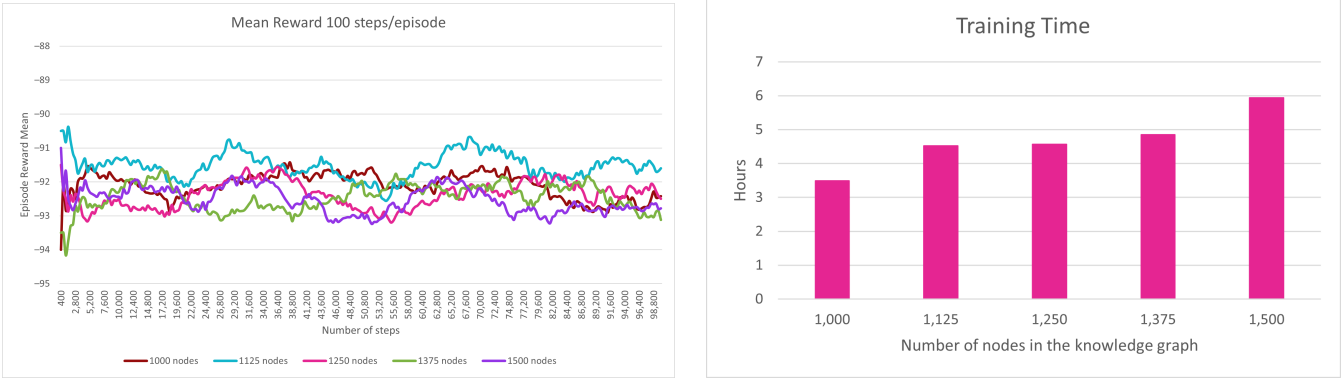
In order to prove the scalability and effectiveness of the solution, several more experiments were conducted on KGs containing larger number of nodes. In particular, the DQN agent's behavior was monitored and evaluated for KGs having 1000, 1125, 1250, 1375, and 1500 nodes, each trained with 100, 250, and 500 steps/episode.

In Figure 10a, it can be observed that the reward is quite unstable given the small number of steps/episode. The configuration that can be visualized in Figure 11a proved to be the best one for larger sizes because of the reward trend. Furthermore, the configuration with 500 steps/episode presented in Figure 12a is improved since for all the KG sizes, besides the one containing 1375 nodes, the reward is increasing and then converging. In

addition, the training time increased linearly with the size of the KG for each number of steps/episode chosen.

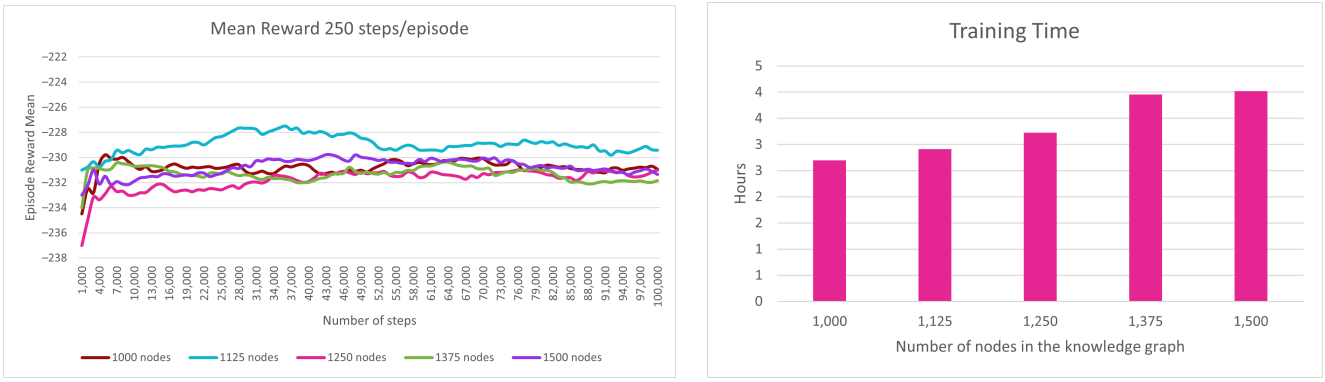
Table 4. Offloading results overview.

Knowledge Graph Size	Number of Steps/Episode	Number of Pairs
Small	100	2
Small	250	9
Small	500	20
Medium	100	3
Medium	250	10
Medium	500	12
Large	100	4
Large	250	16
Large	500	26



(**a**) Mean Reward (**b**) Training time

Figure 10. DQN training behavior for knowledge graphs with more than 1000 nodes and 100 steps/episode.



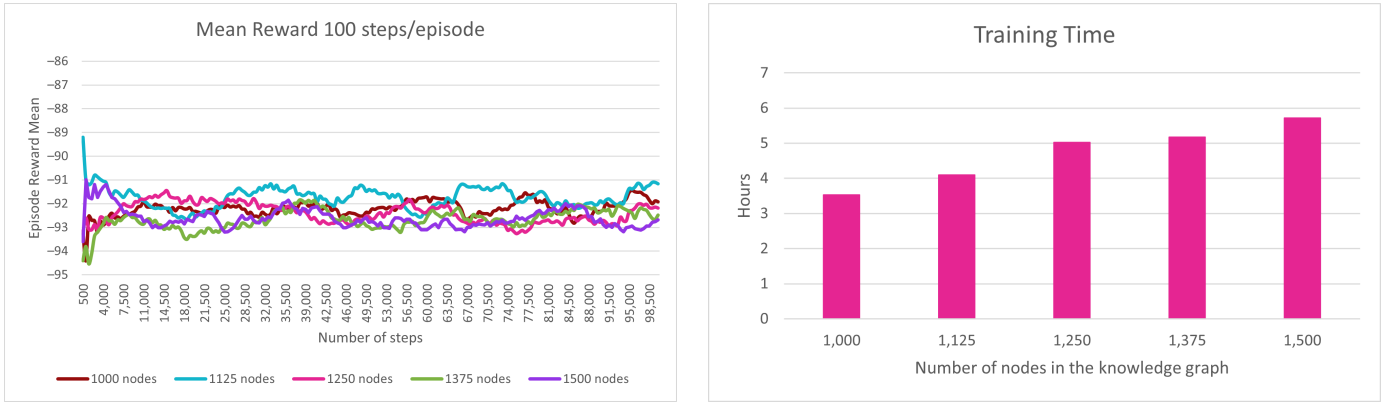
(**a**) Mean Reward (**b**) Training time

Figure 11. DQN training behavior for knowledge graphs with more than 1000 nodes and 250 steps/episode.

In order to explore new RL approaches, several other experiments were conducted on KGs having 1000, 1125, 1250, 1375, and 1500 nodes. For these experiments, the Actor-Critic (A2C) algorithm was chosen and compared with the previous DQN algorithm. In Figures 13–15, the training behavior and training time is presented for the A2C algorithm, having the parameter of 100, 250, or 500 steps/episode. From the figures, it can be observed that the mean reward is not as steady, increasing and converging for the 250 and 500 steps/episode as the one for the DQN approach. In addition, the training time was, in most of the cases, higher than the training time for the DQN. This is because the DQN is more data efficient than A2C.

(**a**) Mean Reward (**b**) Training time

Figure 12. DQN training behavior for knowledge graphs with more than 1000 nodes and 500 steps/episode.

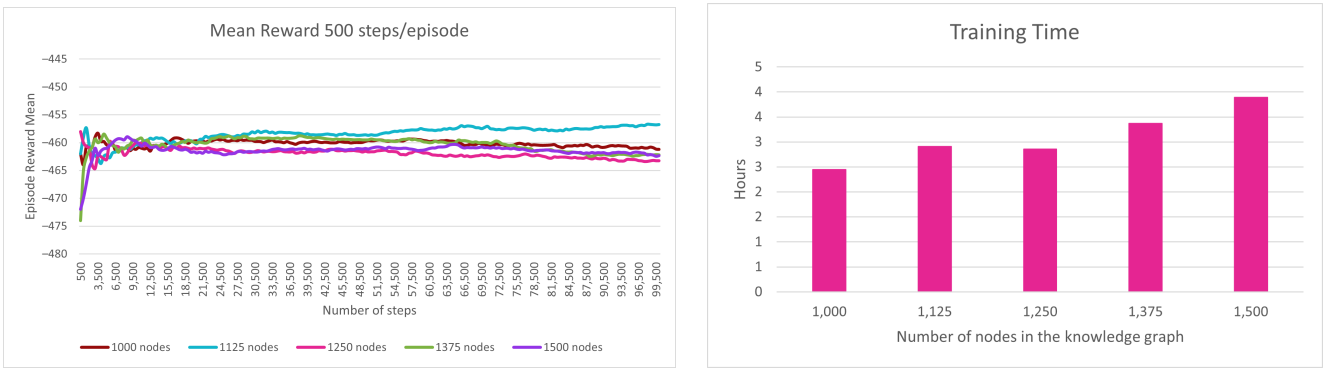


(**a**) Mean Reward (**b**) Training time

Figure 13. A2C training behavior for knowledge graphs with more than 1000 nodes and 100 steps/episode.

(**a**) Mean Reward (**b**) Training time

Figure 14. A2C training behavior for knowledge graphs with more than 1000 nodes and 250 steps/episode.



(**a**) Mean Reward (**b**) Training time

Figure 15. A2C training behavior for knowledge graphs with more than 1000 nodes and 500 steps/episode.

The number of offloading pairs can be consulted in Table 5 for both the DQN and the A2C approaches. The offloading pairs found in the testing episode were made of nodes within the same layer (e.g., fog–fog), but during training, the algorithms were also providing results cross-layer. The A2C method proved to be better for a smaller number of steps/episode, but overall, the DQN performance was better.

Table 5. Offloading results overview (DQN vs. A2C).

Knowledge Graph Size (Nodes)	Number of Steps/Episode	Number of Pairs for DQN	Number of Pairs for A2C
1000	100	1	4
1000	250	11	13
1000	500	21	17
1125	100	4	6
1125	250	13	11
1125	500	24	22
1250	100	2	3
1250	250	13	8
1250	500	17	18
1375	100	7	3
1375	250	16	5
1375	500	14	20
1500	100	3	3
1500	250	10	6
1500	500	23	13

5. Conclusions

In this paper, we defined an efficient task offloading framework using Deep Q-Learning with a custom KG environment representing the computing continuum nodes over the IoT-driven large-scale infrastructures. The edge–fog–cloud infrastructure is modeled as a KG, having nodes with different computational and network characteristics. An adapted version of the Deep Q-Learning algorithm was proposed as the solution for the task offloading to optimize resource usage. The particularity consisted of a custom reward function, based on the node types, sub-types, and unique features. The workload exchange is possible between nodes within the same layer or between nodes from different layer (cloud to fog or fog to edge). During the process, the agent would select pairs of nodes from the KG, verify if they are suitable for the offloading, and then perform the actual offloading between the nodes while updating the KG environment.

The performance of the DQN agent was evaluated using different configurations of the hyperparameters and by varying the episode length or the size of the KG model. A low and steady learning rate and a large buffer size, for the agent's experience, proved to

be key characteristics. The number of pairs of workload tasks and nodes obtained after each learning episode provided insightful information. The number of offloading pairs and actions increases with the size of the KG and the number of steps/episodes.

As for further improvements, we plan to enhance the RL solution by partitioning workloads based on their type characteristics for separate offloading, which will boost efficiency and provide better balance and real-time interaction with data from IoT devices. Moreover, the usage of a distributed deep learning scheduler which divides the KGs would address the potential overhead issues. Furthermore, we aim to move the problem to an online setting, because the addition of the time dimension would be more beneficial to both the users and the system.

Author Contributions: Conceptualization, I.R.M. and T.C.; methodology, I.R.M., T.C. and G.I.A.; software, I.R.M.; validation, I.R.M. and G.I.A.; formal analysis, I.R.M. and T.C.; investigation, G.I.A. and I.R.M.; data curation, I.R.M.; writing—original draft preparation, I.R.M. and T.C.; writing—review and editing, T.C.; visualization, I.R.M.; supervision, T.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been conducted within the HEDGE-IoT project grant number 101136216 funded by the European Commission as part of the Horizon Europe Framework Programme. However, views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Climate, Infrastructure and Environment Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors upon request.

Conflicts of Interest: Author Ms. Ioana Ramona Martin and Mr. Gabriel Ioan Arcas are employed by the company Bosch. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Honar Pajooh, H.; Rashid, M.; Alam, F.; Demidenko, S. IoT Big Data provenance scheme using blockchain on Hadoop ecosystem. *J. Big Data* **2021**, *8*, 114. [CrossRef]
2. Orive, A.; Agirre, A.; Truong, H.L.; Sarachaga, I.; Marcos, M. Quality of Service Aware Orchestration for Cloud–Edge Continuum Applications. *Sensors* **2022**, *22*, 1755. [CrossRef] [PubMed]
3. Kumar, R.; Baughman, M.; Chard, R.; Li, Z.; Babuji, Y.; Foster, I.; Chard, K. Coding the Computing Continuum: Fluid Function Execution in Heterogeneous Computing Environments. In Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 17–21 June 2021; pp. 66–75. [CrossRef]
4. Jansen, M.; Wagner, L.; Trivedi, A.; Iosup, A. Continuum: Automate Infrastructure Deployment and Benchmarking in the Compute Continuum. In Proceedings of the Companion of the 2023 ACM/SPEC International Conference on Performance Engineering, New York, NY, USA, 15–19 April 2023; ICPE '23 Companion; pp. 181–188. [CrossRef]
5. Singh, R.; Gill, S.S. Edge AI: A survey. *Internet Things Cyber-Phys. Syst.* **2023**, *3*, 71–92.
6. Al-Dulaimy, A.; Jansen, M.; Johansson, B.; Trivedi, A.; Iosup, A.; Ashjaei, M.; Galletta, A.; Kimovski, D.; Prodan, R.; Tserpes, K.; et al. The computing continuum: From IoT to the cloud. *Internet Things* **2024**, *27*, 101272. [CrossRef]
7. Eang, C.; Ros, S.; Kang, S.; Song, I.; Tam, P.; Math, S.; Kim, S. Offloading Decision and Resource Allocation in Mobile Edge Computing for Cost and Latency Efficiencies in Real-Time IoT. *Electronics* **2024**, *13*, 1218. [CrossRef]
8. Gabriel Ioan, A.; Cioara, T.; Anghel, I.; Lazea, D.B.; Hangan, A. Edge Offloading in Smart Grid. *Smart Cities* **2024**, *7*, 680–711. [CrossRef]
9. Gabriel Ioan, A.; Cioara, T.; Anghel, I. Whale Optimization for Cloud–Edge-Offloading Decision-Making for Smart Grid Services. *Biomimetics* **2024**, *9*, 302. [CrossRef]
10. Akter, A.; Zafir, E.I.; Dana, N.H.; Joysoyal, R.; Sarker, S.K.; Li, L.; Muyeen, S.M.; Das, S.K.; Kamwa, I. A review on microgrid optimization with meta-heuristic techniques: Scopes, trends and recommendation. *Energy Strategy Rev.* **2024**, *51*, 101298.
11. BahraniPour, F.; Mood, S.E.; Farshi, M. Energy-delay aware request scheduling in hybrid Cloud and Fog computing using improved multi-objective CS algorithm. *Soft Comput.* **2024**, *28*, 4037–4050. [CrossRef]
12. Acheampong, A.; Zhang, Y.; Xu, X.; Kumah, D. A Review of the Current Task Offloading Algorithms, Strategies and Approach in Edge Computing Systems. *Comput. Model. Eng. Sci.* **2022**, *134*, 35–88. [CrossRef]
13. Sun, Z.; Mo, Y.; Yu, C. Graph-Reinforcement-Learning-Based Task Offloading for Multiaccess Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 3138–3150. [CrossRef]

14. Wang, T.; Ouyang, X.; Sun, D.; Chen, Y.; Li, H. Offloading Strategy Based on Graph Neural Reinforcement Learning in Mobile Edge Computing. *Electronics* **2024**, *13*, 2387. [CrossRef]
15. Ntentos, E.; Warnett, S.J.; Zdun, U. Supporting Architectural Decision Making on Training Strategies in Reinforcement Learning Architectures. In Proceedings of the 2024 IEEE 21st International Conference on Software Architecture (ICSA), Hyderabad, India, 4–8 June 2024; pp. 90–100. [CrossRef]
16. Robles-Enciso, A.; Skarmeta, A.F. Adapting Containerized Workloads for the Continuum Computing. *IEEE Access* **2024**, *12*, 104102–104114. [CrossRef]
17. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
18. Phan, L.A.; Nguyen, D.T.; Lee, M.; Park, D.H.; Kim, T. Dynamic fog-to-fog offloading in SDN-based fog computing systems. *Future Gener. Comput. Syst.* **2021**, *117*, 486–497. [CrossRef]
19. Liao, Y.; Shou, L.; Yu, Q.; Ai, Q.; Liu, Q. Joint offloading decision and resource allocation for mobile edge computing enabled networks. *Comput. Commun.* **2020**, *154*, 361–369. [CrossRef]
20. Liu, J.; Zhang, Q. Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality. *IEEE Access* **2019**, *7*, 11222–11236. [CrossRef]
21. Kuang, L.; Gong, T.; OuYang, S.; Gao, H.; Deng, S. Offloading decision methods for multiple users with structured tasks in edge computing for smart cities. *Future Gener. Comput. Syst.* **2020**, *105*, 717–729. [CrossRef]
22. Martin, I.R.; Cioara, T.; Arcas, G.I.; Anghel, I.; Bertoncini, M. Knowledge Graph Model for Computing Continuum over Smart Grid. In Proceedings of the Fifteenth International Conference on Information, Intelligence, Systems and Applications (IISA 2024), Crete, Greece, 17–20 July 2024.
23. Kiss, T.; Ullah, A.; Terstyanszky, G.; Kao, O.; Becker, S.; Verginadis, Y.; Michalas, A.; Stankovski, V.; Kertesz, A.; Ricci, E.; et al. Swarmchestrate: Towards a Fully Decentralised Framework for Orchestrating Applications in the Cloud-to-Edge Continuum. In Proceedings of the Advanced Information Networking and Applications, Kitakyushu, Japan, 17–19 April 2024; Barolli, L., Ed.; Springer: Cham, Switzerland, 2024; pp. 89–100.
24. Su, Q.; Zhang, Q.; Zhang, X. Energy-Aware Cloud-Edge Collaborative Task Offloading with Adjustable Base Station Radii in Smart Cities. *Mathematics* **2022**, *10*, 3992. [CrossRef]
25. Wang, Y.; Jin, X.; Xu, R.; Shao, W.; Lin, F. Task Offloading Based-on Deep Reinforcement Learning for Microgrid. In Proceedings of the 2022 IEEE 10th International Conference on Information, Communication and Networks (ICICN), Zhangye, China, 23–24 August 2022; pp. 281–285. [CrossRef]
26. Nimkar, S.; Khanapurkar, M. Design of a Q-learning based Smart Grid and smart Water scheduling model based on Heterogeneous Task Specific Offloading process. In Proceedings of the 2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON), Bangalore, India, 23–25 December 2022; pp. 1–9. [CrossRef]
27. Zhou, H.; Zhang, Z.; Li, D.; Su, Z. Joint Optimization of Computing Offloading and Service Caching in Edge Computing-Based Smart Grid. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1122–1132. [CrossRef]
28. Wang, T.; Liang, Y.; Zhang, Y.; Zheng, X.; Arif, M.; Wang, J.; Jin, Q. An Intelligent Dynamic Offloading From Cloud to Edge for Smart IoT Systems With Big Data. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 2598–2607. [CrossRef]
29. Mattia, G.P.; Beraldi, R. Online Decentralized Scheduling in Fog Computing for Smart Cities Based on Reinforcement Learning. *IEEE Trans. Cogn. Commun. Netw.* **2024**, *10*, 1551–1565. [CrossRef]
30. Saeed, M.M.; Saeed, R.A.; Mokhtar, R.A.; Khalifa, O.O.; Ahmed, Z.E.; Barakat, M.; Elnaim, A.A. Task Reverse Offloading with Deep Reinforcement Learning in Multi-Access Edge Computing. In Proceedings of the 2023 9th International Conference on Computer and Communication Engineering (ICCCE), Kuala Lumpur, Malaysia, 15–16 August 2023; pp. 322–327. [CrossRef]
31. Basheer, S.; Nalband, A.H. Binary Offloading in Multi-Access Edge Computing Systems: Deep Reinforcement Learning Approach. In Proceedings of the 2024 International Conference on Smart Systems for applications in Electrical Sciences (ICSSES), Tumakuru, India, 3–4 May 2024; pp. 1–6. [CrossRef]
32. Yan, Q.; Ding, M.; Lu, J.; Lu, J.; Yang, H.; Xie, F. Construction of Dynamic Knowledge Graph for Grid day-ahead Scheduling Optimization Decision. In Proceedings of the 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 17–19 June 2022; Volume 10, pp. 1821–1826. [CrossRef]
33. Chun, S.; Jung, J.; Jin, X.; Seo, S.; Lee, K.H. Designing an Integrated Knowledge Graph for Smart Energy Services. *J. Supercomput.* **2020**, *76*, 8058–8085. [CrossRef]
34. Mal, M.; Gong, C.; Zeng, L.; Yang, Y. MOGR: Multi-task Offloading via Graph Representation in Heterogeneous Computing Network. In Proceedings of the ICC 2024—IEEE International Conference on Communications, Denver, CO, USA, 9–13 June 2024; pp. 1237–1242. [CrossRef]
35. Li, J.; Gu, B.; Qin, Z.; Han, Y. Graph Tasks Offloading and Resource Allocation in Multi-Access Edge Computing: A DRL-and-Optimization-Aided Approach. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3707–3718. [CrossRef]
36. Gao, Z.; Yang, L.; Dai, Y. Fast Adaptive Task Offloading and Resource Allocation in Large-Scale MEC Systems via Multiagent Graph Reinforcement Learning. *IEEE Internet Things J.* **2024**, *11*, 758–776. [CrossRef]
37. Li, N.; Iosifidis, A.; Zhang, Q. Graph Reinforcement Learning-based CNN Inference Offloading in Dynamic Edge Computing. In Proceedings of the GLOBECOM 2022—2022 IEEE Global Communications Conference, Rio de Janeiro, Brazil, 4–8 December 2022.

38. Leng, L.; Li, J.; Shi, H.; Zhu, Y. Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing. *Multimed. Tools Appl.* **2021**, *80*, 29163–29175. [CrossRef]
39. Xu, A.; Hu, Z.; Li, X.; Tian, R.; Zhang, X.; Chen, B.; Xiao, H.; Zheng, H.; Feng, X.; Zheng, M.; et al. TransEdge: Task Offloading with GNN and DRL in Edge Computing-Enabled Transportation Systems. *IEEE Internet Things J.* **2024**, *Early Access*. [CrossRef]
40. Liu, J.; Mi, Y.; Zhang, X.; Li, X. Task graph offloading via deep reinforcement learning in mobile edge computing. *Future Gener. Comput. Syst.* **2024**, *158*, 545–555. [CrossRef]