Topology-Aware GNN-DQN for Optimal Task Offloading in Edge–Fog–Cloud Systems

Sirine HAKIM
CY Tech / ENSEA, France
sirine.hakim@ensea.fr

Sonia YASSA
Laboratoire ETIS, UMR 8051
CY Cergy Paris Universités, ENSEA, CNRS
sonia.yassa@cyu.fr

ABSTRACT
The Internet of Things (IoT) has grown rapidly in recent years, enabling the interconnection of a large number of heterogeneous and distributed devices. This number is expected to exceed 70 billion according to Statista. With this massive scale, fulfilling complex IoT applications that require combinations of multiple objects remains a real challenge. Moreover, several Quality of Service (QoS) requirements must be satisfied, making the problem of selecting appropriate IoT services NP-hard. In such environments, task offloading is a key mechanism to efficiently distribute computational workloads across edge, fog, and cloud resources. However, selecting the optimal offloading decision remains a difficult NP-hard problem due to system heterogeneity and conflicting objectives. In this paper, we propose a GNN-DQN-based approach for task offloading in edge–fog–cloud environments. Unlike prior GNN-DQN approaches limited to single- or dual-tier architectures, our framework explicitly models heterogeneous node types and inter-tier communication links, enabling more balanced and scalable resource allocation. Experimental results show that GNN-DQN achieves a mean latency of 2.64 s, representing improvements of 70.2% over Random, 7.8% over DQN-only, and 3.5% over Greedy. A GNN-A2C baseline is also included to broaden the comparison with a modern DRL method. Despite sharing the same GNN encoder, it underperforms GNN-DQN across all metrics, confirming the superiority of the DQN learning backbone. These results highlight the effectiveness of integrating graph-based representation with reinforcement learning, while also revealing a trade-off between latency optimization and energy efficiency.

KEY WORDS: Task offloading, Edge–Fog–Cloud Computing, AI, DQN, GNN, QoS, Decision-Making, Multi-Objective Optimization

I. INTRODUCTION

Over recent years, the Internet of Things (IoT) has become increasingly integrated into daily life, enabling applications such as access control, face recognition, smart cities, smart healthcare, and smart homes [1]. This paradigm aims to ensure seamless connectivity and interactions between people, devices, and data. However, the large scale and heterogeneity of IoT environments make their design and deployment highly challenging. Cloud computing, as defined by NIST, provides scalable storage, processing, and networking resources, but sending large volumes of IoT data to distant cloud data centers may cause high latency and bandwidth overload. To address these limitations, fog and edge computing bring resources closer to data sources, enabling low-latency and real-time services [2].

In this context, task offloading plays a key role in assigning IoT tasks to appropriate computing resources while satisfying Quality of Service (QoS) requirements such as latency and energy consumption. These objectives reflect both user needs for real-time performance and system-level constraints for energy efficiency. More broadly, the integration of artificial intelligence into IoT, referred to as Artificial Intelligence of Things (AIoT), enables systems to make adaptive and intelligent decisions in dynamic environments [1].

Despite the growing interest in deep reinforcement learning (DRL)-based task offloading, current approaches still present important limitations. Many rely on flat feature representations that fail to capture structural relationships between heterogeneous nodes, leading to limited topological awareness [3]. In addition, most existing solutions focus on single-tier or dual-tier architectures, which reduces their applicability in realistic edge–fog–cloud environments [4]. Standard DQN-based methods also face scalability issues as the network size increases, and many studies optimize only one objective, such as latency or energy, instead of jointly considering both [2].

To overcome these limitations, this paper proposes a GNN-DQN-based approach for intelligent task offloading. The GNN captures the structural relationships between nodes, while the DQN learns an effective offloading policy. The proposed method is evaluated against Random, Greedy, DQN-only, and GNN-A2C baselines, showing its effectiveness in improving offloading decisions.

The main contributions of this study can be summarized as follows:

- A three-tier GNN-based edge-fog-cloud infrastructure representation
- A topology-aware offloading framework based on GNN-DQN
- A joint latency–energy multi-objective optimization formulation
- A realistic evaluation using a multi-feature real-world IoT task dataset and a discrete-event simulation of edge–fog–cloud environments

The remainder of this paper is organized as follows: Section II reviews related work, Section III presents the proposed GNN–DQN framework, Section IV describes the experimental setup, Section V reports the results, and Section VI concludes the paper with future directions.

II. Related Work

Existing research on task offloading can be broadly classified into heuristic, metaheuristic, federated learning, machine learning, and hybrid approaches. Heuristic methods, such as the Markov-based solution proposed by Kayal et al. [5], rely on iterative local optimization strategies, while metaheuristic techniques, including the multi-objective evolutionary algorithm introduced by Khadir et al. [6], address QoS optimization problems. Federated learning approaches, such as FedMeta2Ag by Samafou et al. [7], enable distributed task offloading and resource allocation. Machine learning methods, particularly reinforcement learning, have been widely adopted, as illustrated by the DRL-based solution for vehicular fog computing proposed by Toopchinezhad et al. [8]. In parallel, graph-based modeling has been explored to represent service dependencies using Directed Acyclic Graphs (DAGs) [2], and GNN-based frameworks such as *Fograph* have demonstrated significant improvements in latency and throughput in distributed environments [9]. Finally, hybrid approaches combining multiple techniques, such as reinforcement learning and NOMA, have been proposed by Al-Abbasi et al. [3] to enhance resource allocation and offloading decisions.

Despite these advances, key limitations remain: heuristic and metaheuristic methods lack scalability, federated learning introduces communication overhead, standard RL relies on flat representations that miss network topology, and most approaches optimize a single QoS metric. Combining GNN with DQN directly addresses these gaps by enabling topology-aware representations and joint QoS optimization in large-scale edge–fog–cloud environments.

III. Topology-Aware GNN–DQN Framework for Task Offloading

A. System Model and Problem Formulation

The task offloading problem in edge–fog–cloud computing consists of assigning IoT tasks to a set of candidate node in order to optimize overall Quality of Service (QoS). In this work, we consider latency and energy consumption as QoS metrics.

IoT tasks are represented as a set:

$$T = \{t_1, t_2, \ldots, t_n\} \tag{1}$$

Each task t_i is defined as:

$$t_i = (S_i, C_i, T_i^{gen}, R_i, D_i) \tag{2}$$

where:

- S_i: task size (bits)
- C_i: CPU cycles per bit
- T_i^{gen}: generation time
- R_i: transmission rate (bits/s)
- D_i: deadline (s)

B. Topology-Aware GNN–DQN Architecture

This paper proposes a GNN-DQN framework that encodes the network topology as a graph to capture relational dependencies between nodes and enabling more accurate and scalable offloading decisions across the three-tier edge-fog-cloud architecture.

The edge–fog–cloud infrastructure is modeled as a graph:

$$G = (V, E) \tag{3}$$

where $V = \{v_1, v_2, ..., v_N\}$ denotes computing nodes (edge, fog, cloud) and $E \subseteq V \times V$ denotes communication links.

The Deep Q-Network (DQN) approximates the action-value function $Q(s, a)$.

Given a state s_t, the selected action is:

$$a_t = \arg\max_{a \in A} Q(s_t, a; \theta_Q) \tag{4}$$

The Q-network is trained by minimizing the loss:

$$L(\theta_Q) = \mathbb{E}\left[\left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_Q^-) - Q(s_t, a_t; \theta_Q)\right)^2\right] \tag{5}$$

where r_t is the reward, $\gamma \in [0, 1]$ is the discount factor, and θ_Q^- denotes the target network parameters.

The objective is to maximize cumulative reward over time.

1) GNN-DQN Workflow

First, the edge–fog–cloud infrastructure is modeled as a graph to capture relationships between computing nodes.

A Graph Neural Network (GNN) is used to extract embeddings:

$$h_t = \text{GNN}(G; \theta_{GNN}) \tag{6}$$

The state is defined as:

$$s_t = [h_t \parallel x_t] \tag{7}$$

where x_t represents task features and h_t represents the graph embedding.

The DQN then selects the optimal offloading action based on s_t, and the reward is computed using latency and energy consumption.

Experience replay is used to train the network.

C. QoS Modeling: Latency, Energy and Optimization

We define $x_{ij} \in \{0, 1\}$ as the task offloading decision variable, indicating whether task i is assigned to node j. The objective is to jointly minimize latency and energy consumption under computational and communication constraints.

1) Latency Model

The total latency is defined as:

$$L(i, j) = T_{\text{trans}}(i, j) + T_{\text{wait}}(i, j) + T_{\text{exec}}(i, j) \tag{8}$$

The transmission delay is defined as:

$$T_{\text{trans}}(i, j) = \sum_{\ell \in P_{ij}} b_\ell + \frac{S_i}{R_{ij}} \tag{9}$$

where P_{ij} denotes the communication path, b_ℓ is the propagation delay of link ℓ, S_i is the task size, and R_{ij} is the transmission rate.

The execution delay is defined as:

$$T_{\text{exec}}(i, j) = \frac{S_i \cdot C_i}{f_j} \tag{10}$$

where C_i is the number of CPU cycles per bit and f_j is the processing capacity of node j. The term $T_{\text{wait}}(i,j)$ represents the queueing delay at node j.

2) Energy Model

The total energy consumption is defined as

$$E(i,j) = E_{\text{exec}}(i,j) + E_{\text{trans}}(i,j) \tag{11}$$

The execution energy is modeled as

$$E_{\text{exec}}(i,j) = P_j^{\text{exec}} \cdot T_{\text{exec}}(i,j) \tag{12}$$

where P_j^{exec} denotes the active power consumption of node j, while the transmission energy is defined as

$$E_{\text{trans}}(i,j) = \sum_{\ell \in P_{ij}} \kappa_\ell \cdot S_i \tag{13}$$

where κ_ℓ is the energy coefficient of link ℓ.

3) Reward Function

The reinforcement learning reward is defined as

$$R = -(\alpha L(i,j) + \beta E(i,j)) \tag{14}$$

which reflects the trade-off between latency and energy consumption.

D. Multi-Objective Optimization Formulation

The task offloading problem is formulated as

$$\min \sum_i \sum_j x_{ij}(\alpha L(i,j) + \beta E(i,j)) \tag{15}$$

subject to:

- $\sum_j x_{ij} = 1, \ \forall i$
- CPU capacity constraints at each node
- memory and bandwidth limitations
- task execution feasibility constraints

The action space is defined as

$$A = \{1, 2, \ldots, N\} \tag{16}$$

E. Learning-Based Offloading Algorithm

The proposed GNN-DQN algorithm is described in Algorithm 1. For each epoch, the environment is reset, and for every incoming task, the GNN first encodes the edge-fog-cloud graph to produce topological embeddings h_t. These embeddings are concatenated with task features x_t to form the state s_t, which is passed to the DQN. The agent selects an offloading action a_t using an ε-greedy policy, executes it, and observes the resulting latency L_t and energy E_t, which are combined into a reward r_t. The transition (s_t, a_t, r_t, s_{t+1}) is stored in a replay buffer $\mathcal{D}$. A mini-batch is sampled to update the Q-network by minimizing the temporal difference loss, while a target network θ_Q^- periodically synchronizes to stabilize learning. The process repeats with decaying exploration ε, ultimately converging to a near-optimal offloading policy π^*.

Algorithm 1 GNN-DQN for Task Offloading

Input: Graph $G = (V, E)$, task stream $\{\tau_1, \ldots, \tau_T\}$
Output: Policy π^*
Initialize $\theta_{\text{GNN}}, \theta_Q$
Initialize target network $\theta_Q^- \leftarrow \theta_Q$
Initialize replay buffer $\mathcal{D}$
for epoch $= 1$ to M **do**
  Reset environment
  for each task τ_t **do**
    Observe task features x_t
    $h_t \leftarrow \text{GNN}(G; \theta_{\text{GNN}})$
    $s_t \leftarrow [h_t \,\|\, x_t]$
    Select a_t using ε-greedy from $Q(s_t, a; \theta_Q)$
    Execute action a_t
    Observe latency L_t and energy E_t
    $r_t \leftarrow -(\alpha L_t + \beta E_t)$
    Observe next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample mini-batch from $\mathcal{D}$
    Update θ_Q using:

$$\mathcal{L} = \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta_Q^-) - Q(s_t, a_t; \theta_Q) \right)^2$$

    if update step **then**
      $\theta_Q^- \leftarrow \theta_Q$
    end if
  end for
  Decay ε
end for
return $\pi^*(s) = \arg\max_a Q(s, a; \theta_Q)$

F. Computational Complexity

The computational complexity of the proposed GNN-DQN framework arises from two main components. The GNN encoder processes the graph with a complexity of $\mathcal{O}(L(|V| + |E|))$, where L is the number of layers, and $|V|$ and $|E|$ are the numbers of nodes and edges. This scales linearly with graph size. The DQN training adds a complexity of $\mathcal{O}(B \cdot d \cdot A)$, with B the batch size, d the state dimension, and A the number of actions. Overall, the framework scales near-linearly with infrastructure size, and the GNN encoding is parallelizable while DQN inference remains lightweight, supporting real-time offloading decisions.

IV. Experimental Setup and Evaluation Protocol

A. Dataset and Simulation Environment

In this study, we use the Pakistan dataset, previously validated in task offloading research [10]. This dataset was selected because it captures realistic IoT task diversity across multiple device types and data categories, and has been previously validated in task offloading research. Its nine features (TaskID, TaskName, GenerationTime T_i^{gen}, TaskSize S_i, CyclesPerBit C_i, TransmissionBitRate R_i, Deadline D_i, DataType, DeviceType) directly map to the QoS parameters of our system model, making it well-suited for evaluating latency- and energy-aware offloading policies in heterogeneous edge–fog–cloud environments.

To evaluate the proposed model, a simulation framework was developed to mimic real-world edge-fog-cloud environments. RayCloudSim, a Python-based discrete-event simulator, is used to model multi-tier edge–fog–cloud architectures and evaluate task offloading scenarios. It enables the evaluation of

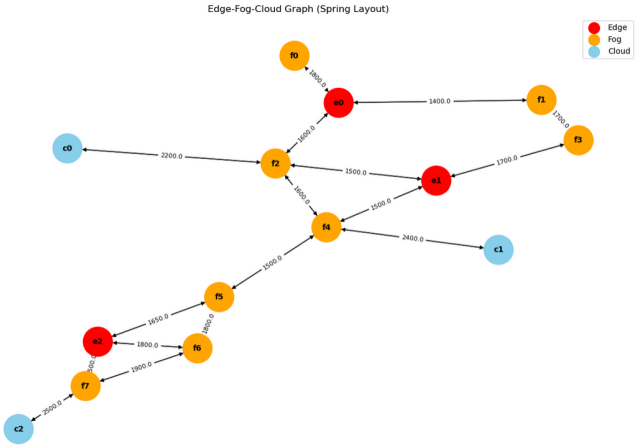

Fig. 1. Edge–Fog–Cloud Network Graph

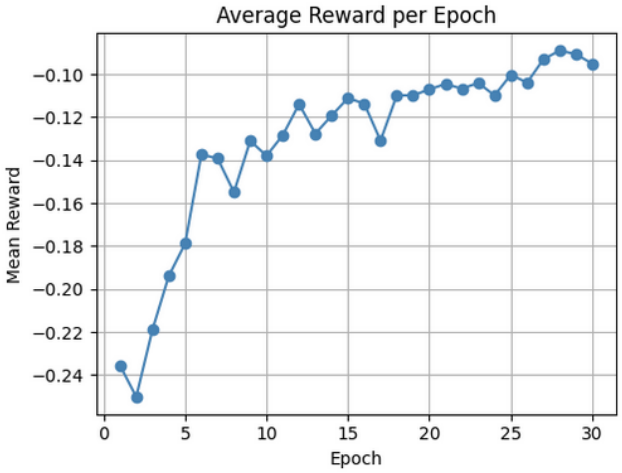

Fig. 2. Average Reward per Epoch

resource management, scheduling, and task offloading policies under heterogeneous infrastructure and network conditions.

B. Model Architecture and Training Configuration

1) GNN Encoding and Graph Representation

The GNN, based on a GCN architecture, models the edge–fog–cloud infrastructure as a graph, as shown in Fig. 1, of 14 nodes and 17 edges, where nodes represent computing resources (edge, fog, and cloud servers) and edges represent communication links between them, weighted by bandwidth capacity. Three node types are considered: edge (red), fog (orange), and cloud (blue), each characterized by CPU capacity, buffer size, and energy coefficients. Three convolutional layers (hidden size 64, ReLU, dropout 0.1) learn node embeddings that capture both topology and available resources.

2) DQN implementation

The Q-network consists of two hidden layers with 256 and 128 neurons using ReLU activation and a dropout rate of 0.1. It outputs a single Q-value per node. The model is trained with Adam (learning rate 3×10^{-4}) and SmoothL1Loss, using a discount factor of 0.99, a batch size of 32, and a replay buffer of 20,000. The target network is updated every 50 steps, and an epsilon-greedy strategy is applied, with epsilon decreasing from 1.0 to 0.05. Training is performed over 30 epochs on up to 1,000 tasks.

3) GNN-DQN training

The training follows a GNN-DQN framework over 30 epochs with up to 1000 tasks per epoch. For each task, node and graph embeddings are recomputed using the GNN to capture task-aware representations. The state combines task features and embeddings, and actions are selected using an epsilon-greedy policy. Rewards are based on latency and energy consumption. Transitions are stored in a replay buffer and used to update the Q-network, while a target network is periodically synchronized and epsilon decays over time to balance exploration and exploitation.

4) Baseline Methods and Evaluation Setup

The proposed GNN-DQN model is evaluated against four baseline methods: a Random strategy, a Greedy approach that selects the best node based on immediate reward, a DQN-only model without GNN to assess the contribution of graph-based representation, and a GNN-A2C model that combines the same GNN encoder with an Advantage Actor-Critic algorithm. DQN-only serves as an ablation baseline to isolate the impact of the GNN component, while GNN-A2C isolates the impact of the learning algorithm independently of the graph encoding. The multi-objective reward weights are set to $\alpha = 0.7$ and $\beta = 0.3$, deliberately emphasizing latency minimization, as meeting task deadlines is the primary QoS requirement in real-time IoT applications. The evaluation is performed over 30 epochs using 500 tasks, ensuring a fair comparison across all methods.

V. Results and Performance Evaluation

A. Training Dynamics and Convergence Analysis

1) Average Reward per Epoch

As shown in Fig. 2, the curve shows a global increasing trend of the average reward across epochs. At the beginning (epochs 1–5), the reward is very low ≈ -0.25, which indicates poor decisions due to high exploration. Between epochs 5 and 15, the reward improves significantly, showing that the agent learns better offloading strategies. Some fluctuations appear, which is normal in reinforcement learning. After epoch 15, the curve becomes more stable and slowly increases. This indicates that the model is converging toward a stable policy.

2) Training Loss

Fig. 3 shows that the training loss initially is high and noisy due to exploration and unstable learning, then gradually decreases as the model improves its Q-value estimations. Despite normal fluctuations from stochasticity, the loss becomes lower and more stable after $\approx 20{,}000$ steps, indicating convergence. This trend confirms that the GNN-DQN training is effective and learns a consistent, near-optimal policy.

3) Epsilon Schedule (per epoch)

As shown in Fig. 4, the epsilon value decreases rapidly during the first epochs, starting from a high value ($\approx$ 0.8). This indicates that the agent initially performs strong exploration, trying many different actions. Between epochs 5 and 12,

Fig. 3. GNN-DQN Training Loss

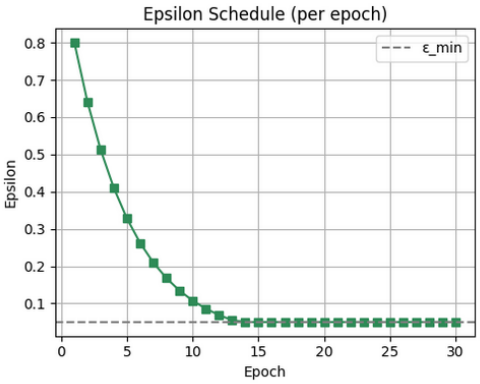

Fig. 4. Epsilon Decay

epsilon continues to decrease smoothly, reducing exploration and increasing exploitation. After around epoch 12–15, epsilon reaches its minimum value and becomes stable. This means the agent now mainly relies on the learned policy instead of random actions. The epsilon schedule is well designed, ensuring a good balance between exploration and exploitation, which is essential for effective learning in GNN-DQN.

B. Comparative Performance Evaluation

1) Mean Reward Comparison

Fig. 5 presents the mean reward comparison across all evaluated policies over 500 test tasks. GNN-DQN, DQN-only, and Greedy achieve comparable mean rewards of -0.0633, -0.0622, and -0.0618 respectively, all significantly outperforming GNN-A2C (-0.1658) and the Random baseline (-0.2681). GNN-A2C exhibits lower performance and higher variance despite sharing the same GNN encoder, while the three intelligent policies show narrow confidence intervals, confirming stable and consistent decision-making. These results confirm that GNN-DQN achieves near-optimal reward performance, closely matching Greedy while operating without exhaustive node simulation.

2) CDF (Cumulative Distribution Function) of Rewards

Fig. 6 presents the CDF of rewards in the range $[-0.175, 0.025]$. GNN-DQN closely matches Greedy, with identical median rewards (-0.062), confirming near optimal performance across the full distribution. DQN-only shows a slight bias in upper quantiles due to the lack of topology-aware embeddings, leading to less balanced decisions. GNN-A2C exhibits a shifted median (-0.100) with a wider distribution,

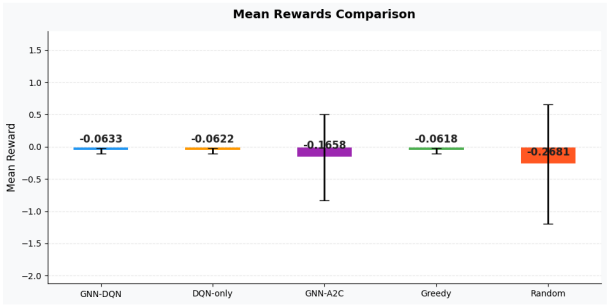

Fig. 5. Mean Reward Comparison between our proposed model, GNN-A2C, and Baselines

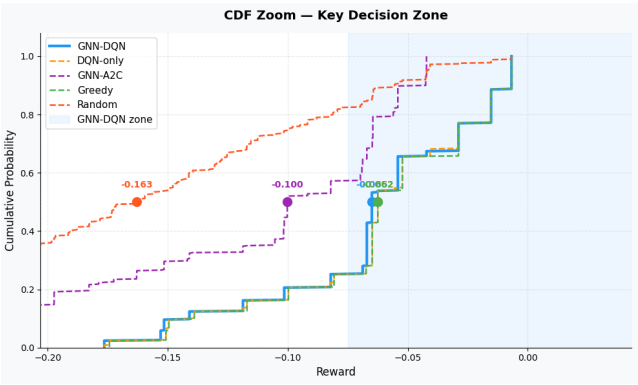

Fig. 6. CDF Rewards

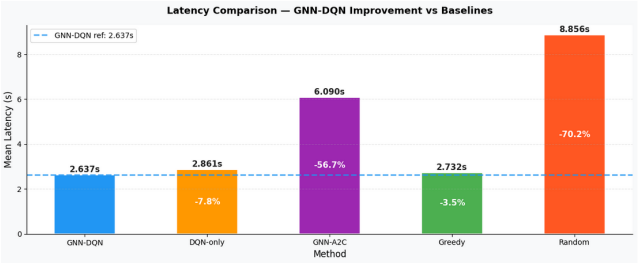

Fig. 7. Mean Latency improved by GNN-DQN

reflecting less consistent offloading decisions. In contrast, Random performs significantly worse (median -0.163), highlighting the importance of learned offloading strategies.

3) Mean Latency Comparison

Fig. 7 compares the mean latency achieved by all methods. GNN-DQN consistently achieves the lowest latency (2.637 s), outperforming Greedy (2.732 s, $+3.5\%$), DQN-only (2.861 s, $+7.8\%$), GNN-A2C (6.090 s, $+56.7\%$), and Random (8.856 s, $+70.2\%$). The large gap between GNN-DQN and GNN-A2C, despite sharing the same encoder, highlights that the learning algorithm plays a decisive role in achieving optimal latency, beyond topology-aware graph encoding alone.

4) Mean Energy Comparison

As shown in Fig. 8, GNN-DQN consumes 98.97 J, slightly more than Greedy (85.87 J) and DQN-only (87.47 J), which is a direct consequence of the reward weights $\alpha = 0.7$ and $\beta = 0.3$ deliberately prioritizing latency over energy efficiency. GNN-A2C consumes 139.67 J, significantly higher than GNN-DQN ($+41.1\%$), confirming that its less controlled policy gradient updates lead to suboptimal node assignments

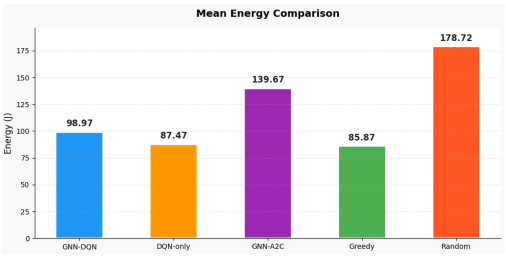

Fig. 8. Mean Energy Comparison

with higher energy overhead. Random performs worst at 178.72 J, consistent with its uninformed allocation behavior. Importantly, when evaluated on the combined objective $\alpha \cdot L + \beta \cdot E$, GNN-DQN achieves the best overall score across all methods, demonstrating that its moderate energy consumption is offset by its superior latency performance, yielding the most favorable multi-objective trade-off.

C. Offloading Policy Analysis Across Computing Tiers

Fig. 9 shows that fog nodes dominate task assignments, representing 67.4%, 81.2%, and 65.6% for GNN-DQN, DQN-only, and Greedy, respectively. GNN-DQN yields a balanced Edge–Fog distribution close to Greedy, indicating near-optimal decisions. By contrast, DQN-only is more strongly biased toward fog nodes, reflecting limited use of edge resources without graph-based representations. GNN-A2C assigns 79.0% of tasks to fog nodes and 14.0% to cloud, the highest cloud usage among intelligent methods, which explains its elevated latency and energy consumption. Cloud nodes are mostly avoided by intelligent methods due to higher latency and energy cost, while Random assigns 20% of tasks to Cloud. This highlights the benefit of GNNs for more balanced and efficient task allocation.

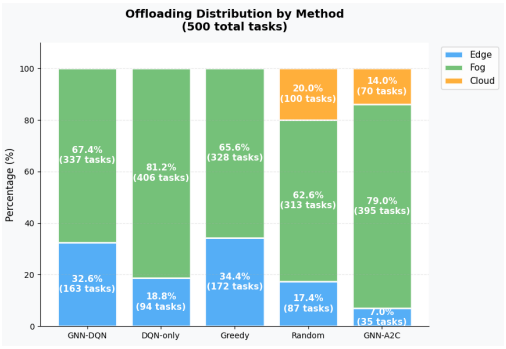

Fig. 9. Task Distribution Across Edge, Fog, and Cloud Nodes

VI. Conclusion and Future Work

This paper demonstrates that topology-aware graph representations, when combined with deep reinforcement learning, enable more balanced and efficient task offloading decisions across heterogeneous edge–fog–cloud infrastructures. The proposed GNN-DQN achieves the best latency performance (2.64 s), outperforming Random, DQN-only, and Greedy by 70.2%, 7.8%, and 3.5% respectively, while maintaining the best combined multi-objective score. Beyond these results, the comparative analysis revealed two key insights. First, GNN-A2C, despite sharing the identical encoder, consistently underperforms GNN-DQN across all metrics, demonstrating that the choice of learning backbone is as critical as topological awareness. Second, topology-aware embeddings naturally promote a balanced edge–fog distribution (32.6% edge, 67.4% fog), reducing the over-reliance on fog nodes observed in flat representations (18.8% edge, 81.2% fog for DQN-only). The observed energy trade-off reflects an inherent tension between latency and energy objectives, suggesting that fixed reward weighting is a limiting factor in multi-objective offloading. Future work will focus on adaptive weighting strategies, more expressive GNN architectures, and evaluation on larger and more diverse topologies to assess scalability in realistic AIoT deployments.

References

[1] H. Gu, L. Zhao, Z. Han, G. Zheng, and S. Song, "AI-Enhanced Cloud-Edge-Terminal Collaborative Network: Survey, Applications, and Future Directions," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 2, pp. 1322–1385, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10336879

[2] I. Taleb, J.-L. Guillaume, and B. Duthil, "A Survey on Services Placement Algorithms in Integrated Cloud-Fog / Edge Computing," *ACM Computing Surveys*, vol. 57, no. 11, pp. 1–36, Nov. 2025. [Online]. Available: https://dl.acm.org/doi/10.1145/3729214

[3] Z. Q. Al-Abbasi, K. M. Rabie, X. Li, W. U. Khan, and A. A. Samah, "RL-Based Adaptive Task-Offloading in Mobile-Edge Computing for IoT Networks," *IEEE Internet of Things Magazine*, pp. 1–7, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11267502

[4] A. Garmendia-Orbegozo, J. D. Nunez-Gonzalez, and M. A. Anton, "Task Offloading in Edge Computing Using GNNs and DQN," *Computer Modeling in Engineering & Sciences*, vol. 139, no. 3, pp. 2649–2671, 2024. [Online]. Available: https://www.techscience.com/CMES/v139n3/55629

[5] P. Kayal and J. Liebeherr, "Autonomic Service Placement in Fog Computing," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. Washington, DC, USA: IEEE, Jun. 2019, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/8792989/

[6] K. Khadir, N. Guermouche, A. Guittoum, and T. Monteil, "A Genetic Algorithm-Based Approach for Fluctuating QoS Aware Selection of IoT Services," *IEEE Access*, vol. 10, pp. 17 946–17 965, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9705594/

[7] F. Samafou, B. Amine Adoum, A. A. Abba Ari, F. Marius Fidel, A. Moungache, N. Armi, and A. Mourad Gueroui, "A new approach to joint resource management in MEC-IoT based federated meta-learning," *Bulletin of Electrical Engineering and Informatics*, vol. 13, no. 5, pp. 3196–3217, Oct. 2024. [Online]. Available: https://beei.org/index.php/EEI/article/view/7993

[8] M. P. Toopchinezhad and M. Ahmadi, "Deep Reinforcement Learning for Delay-Optimized Task Offloading in Vehicular Fog Computing," in *2025 29th International Computer Conference, Computer Society of Iran (CSICC)*, Feb. 2025, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10967415

[9] L. Zeng, X. Chen, P. Huang, K. Luo, X. Zhang, and Z. Zhou, "Serving Graph Neural Networks With Distributed Fog Servers for Smart IoT Services," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 550–565, Feb. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10189178/

[10] M. Aazam, S. u. Islam, S. T. Lone, and A. Abbas, "Cloud of Things (CoT): Cloud-Fog-IoT Task Offloading for Sustainable Internet of Things," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 01, pp. 87–98, Jan. 2022. [Online]. Available: https://www.computer.org/csdl/journal/su/2022/01/09214500/1nHNyGv1j8s