Task Offloading in Vehicular Edge Computing Using Deep Reinforcement Learning

by

Askar Madiyev

Submitted to the Department of Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

NAZARBAYEV UNIVERSITY

April 2025

© Nazarbayev University 2025. All rights reserved.

Author .....
Department of Computer Science
11

Certified by .....
Dr. Meiram Murzabulatov
Professor
Thesis Supervisor

Certified by .....
Dr. Dinh Mao Bui
Professor
Thesis Supervisor

Accepted by .....
Yelizaveta Arkhangelsky
Dean, School of Engineering and Digital Sciences

Task Offloading in Vehicular Edge Computing Using Deep Reinforcement Learning

by

Askar Madiyev

Submitted to the Department of Computer Science
on 11, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Vehicular Edge Computing (VEC), a subset of Mobile Edge Computing (MEC), is a promising technology for real-time processing in vehicular networks at the edge of the network. By shifting computationally demanding activities to one-hop-away edge servers, it facilitates the emergence of applications like intelligent transportation systems and autonomous vehicles [3]. There is a combination of problems within the current VEC research area, including a dynamic environment of moving subjects, application task dependency, limited resources on the edge, and constraints on energy efficiency and end-to-end latency. Conventional offloading techniques, like heuristic methods, can not adequately make decisions in such environments in a reasonable time. The goal of this project is to address these challenges by proposing and comparing a standard Feedforward Deep Q-Network (FF-DQN) and an innovative Transformer-Encoder Double Deep Q-Network (TE-DDQN). The proposed methodology incorporates real-world data measurements from the Nvidia Jetson Nano. This work contributes to the advancement of energy and latency-efficient task offloading strategies for the increasingly important domain of vehicular edge computing.

Thesis Supervisor: Dr. Meiram Murzabulatov
Title: Professor

Thesis Supervisor: Dr. Dinh Mao Bui
Title: Professor

Acknowledgments

This Master Thesis has been funded by the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (Grant No. AP19679952) and by the Faculty Development Competitive Research Grant Program (Funder Project Reference: 021220FD1451), funded by Nazarbayev University, Kazakhstan.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 Contribution 14

2 Literature Review **17**

2.1 Related Works 17

2.1.1 Latency as the Optimization Target 17

2.1.2 Energy as the Optimization Target 19

2.1.3 Latency and Energy as the Optimization Target 20

2.2 Development Environment 21

2.2.1 Software section 21

2.2.2 Hardware section 24

3 Methodology **25**

3.0.1 Energy-Aware Vehicular Edge Computing System 25

3.0.2 Base Station Placement Using K-means Clustering 25

3.0.3 Real-World Energy Consumption Model 26

3.0.4 Simulated Latency Model 32

3.0.5 Problem Formulation 35

3.0.6 Solution Implementation 37

4 Results **45**

4.1 Training Performance Analysis 45

4.1.1 FF DQN training results 45
4.1.2 TE DDQN training results 46
4.2 Comparative Analysis 49
4.2.1 Hyperparameters and Training Configuration 49
4.2.2 Verification by Simulation 50
4.2.3 Physical Testbed Evaluation 57

5 Conclusion **63**
5.1 Limitations and Future Work 63
5.2 Conclusion 64

List of Figures

2-1 Architecture of VEC workflow. 22
2-2 SUMO map config for Astana city. 23

3-1 Base Station Locations. 26

4-1 performance comparison 46
4-2 Training Reward Convergence 47
4-3 Training Trask Completion Rate 47
4-4 Training Energy Consumption 48
4-5 Training Latency 48
4-6 Reward fluctuation FF-DQN vs TE-DDQN 51
4-7 Node load distribution comparison 51
4-8 Performance of multiple models under Low load 52
4-9 Reward distribution of multiple models under Low load 52
4-10 Performance of multiple models under Medium load 53
4-11 Performance of multiple models under Medium load 53
4-12 Performance of multiple models under High load 54
4-13 Performance of multiple models under High load 54
4-14 Arrival rate of tasks as Poisson process 58
4-15 TE-DDQN training results 59
4-16 FF-DQN training results 59
4-17 Simulation results for physical testbed models 59
4-18 Energy consumption results on physical testbed 60
4-19 Average free memory results on physical testbed 60

4-20 Summary of physical testbed results 61

List of Tables

2.1 Objective Evaluation Methods in Recent Works 18

Chapter 1

Introduction

1.1 Motivation

The growth of connected and autonomous vehicles has greatly increased the volume of data produced by vehicular networks, bringing new issues and opportunities for efficient data processing and management [3]. Vehicular Edge Computing (VEC) is an application of Mobile Edge Computing (MEC) to vehicular networks, where computation is offloaded to on-hop away servers from User Equipment (UE) [12]. This approach is an alternative to distant cloud servers that despite having more computational and storage resources lack end-to-end latency, which is essential for applications such as autonomous driving, real-time traffic management, and augmented reality [3]. VEC environment introduces unique patterns not present in standard MEC. Including the mobility of UE which introduces special network conditions, varying signal strengths, crucial factor otight computation deadlines and more strict constraints on resources on edge nodes. In combination those factors make VEC one of the most heteregeneous environment where Deep Learning can be applied.

In VEC environments, efficient task offloading is essential for minimizing end-to-end latency and energy consumption. Traditional offloading strategies, often based on heuristic algorithms or static models like in [12], are limited by their lack of adaptability to dynamic environments. As demonstrated in [24] studies on energy-efficient offloading leveraging Deep Reinforcement Learning (DRL) models can optimize of-

floading decisions utilizing data generated by the environment. Therefore, DRL-based models are more likely to adapt to changing network conditions. However, existing DRL approaches, like frameworks from [24] that are based on the combination of the Genetic Algorithm and Deep Neural Networks (DNN), face challenges with scalability and adaptability when confronted with new network topologies. Because they don't account for the changes in the position of the offloading subjects. Unlike [24], [3] does access the mobility of the subject. However, like most DRL-based solutions, it requires loading the pre-trained or online training DRL model from the memory for each request, thus increasing response time and decreasing Quality of Service (QoS).

1.2 Contribution

This research aims to solve offloading decisions in VEC environments to minimize energy consumption and end-to-end latency under a dynamic vehicular network environment. The problem is expanded to choose between different edge nodes within each server to distribute load and check for the availability of the resources. Also, to manage the resources of the edge nodes more effectively in terms of energy efficiency it is crucial to consider that many applications consist of several interconnected tasks that can be offloaded individually [26]. For instance,[27] proposes a solution which utilizes graph neural networks for task offloading built upon DRL. Their solution is called a multi-task learning-based feedforward neural network (MTFNN) which utilizes a decision data table populated by a continuously trained DRL model [27].

The proposed optimization framework leverages a DRL approach to optimize dynamic task offloading in VEC. It introduces and compares two reinforcement learning solutions: a standard Feedforward Neural Network Deep Q-Network (FF-DQN) and an innovative Transformer-Encoder Double Deep Q-Network (TE-DDQN). The DRL model is trained to learn optimal policies by interacting with the vehicular environment, leveraging real-time inputs such as network states, vehicular mobility, and resource availability. The proposed methodology incorporates real-world energy measurements from Jetson Nano devices and utilizes realistic vehicular mobility patterns

simulated with SUMO using an Astana city map. It provides a more accurate representation of the real-world VEC environment. Also, the integration of Wake-On-Lan capabilites of Jetson Nano allows more granular and feasible control over both latency and energy consumption of the VEC edge nodes. The major objectives of this study are:

- To design and implement a FeedForward and Transformer-based RL approach for energy and latency-efficient task offloading
- To incorporate real-world energy measurements and realistic mobility patterns into the training environment
- To implement dynamic node management with Wake-on-LAN capability for improved energy efficiency
- To compare the performance of standard DQN and DDQN approaches in terms of convergence speed, stability, and overall performance
- To compare the performance of the proposed solution with some baseline techniques.

This document is organized as follows: The review of existing studies and current research trends are discussed in Section 2. The proposed methodology is presented in Section 3. The results are analyzed and presented in Section 4. The future work directions as well as Conclusion of the proposed framework are summarized in Section 5.

Chapter 2

Literature Review

2.1 Related Works

This section examines recent findings in the MEC research field, pinpointing current trends, accepted practices and unfilled gaps. To organize the review the literature was divided into 3 categories based on the optimization target they are addressing: latency as the optimization target, energy as the optimization target, and latency and energy as the optimization target. This separation enables a focused examination of the solutions in each field, highlighting accepted practices and the overall coverage of each optimization target. Furthermore, the literature is assessed according to the kinds of user equipment that are considered, the methods used for verification, and the degree to which solutions can function well in real-world or heterogeneous environments.

2.1.1 Latency as the Optimization Target

The focus on the optimization of latency has been the most well-researched and the primary objective in the research of MEC offloading. A lot of research works succeeded in latency optimization using various approaches such as those proposed by Lin et al. (2020) [4]. This work proposed a Particle Swarm Optimization (PSO) to reduce the latency of task computation on MEC servers. Another approach in

Table 2.1: Objective Evaluation Methods in Recent Works

Literature	**Latency**	**Energy**	**Subject**	**# Subjects**	**Verification Methods**	
					Simulation	**Test bed**
[4]	✓	✗	G	S	✓	✗
[25]	✓	✗	G	M	✓	✗
[17]	✓	✗	G	M	✓	✗
[10]	✓	✗	V	M	✓	✗
[11]	✓	✗	G	S	✗	✓
[7]	✓	✗	I	S	✗	✗
[16]	✓	✗	G	M	✓	✗
[6]	✓	✗	G	M	✓	✗
[20]	✗	✓	V	M	✓	✗
[15]	✗	✓	G	M	✓	✗
[18]	✗	✓	G	S	✓	✗
[13]	✗	✓	G	S	✗	✓
[8]	✗	✓	D	M	✓	✗
[9]	✓	✓	G	S	✓	✗
[14]	✓	✓	I	S	✓	✗
[21]	✓	✓	G	M	✓	✗
[19]	✓	✓	A	M	✗	✓
[5]	✓	✓	D	S	✗	✗
Proposed work	✓	✓	V	M	✓	✓

Legend: G - General User Equipment, V - Vehicles, I - IoT Devices, D - DNN Inference, A - AR application, S - Single, M - Multi

that field is to utilize DRL as the decision engine. For instance, [25] proposed a solution leveraging DRL with long short-term memory (LSTM) network to reduce latency in MEC. Further work with a similar goal and approach was suggested by Do-Duy et al. (2022) [17] which employed Double DQN to improve the performance. However, their choice of more generalised User Equipment (UE) left unanswered challenges introduced by Vehicles in MEC such as mobility. Meanwhile, unlike their predecessors, [10] offered a solution targeting Vehicles as the subject of its Two-layered DRL technique. Although this technique addressed the issues brought by Vehicles to MEC, it neglected energy efficiency for its consideration making it less applicable for real-world situations. Another crucial aspect of the proposed techniques is the evaluation methods applied to verify the applicability of the proposition.

Solutions proposed in [11], [7] applied theoretical verification to their solutions due to their algorithmic nature. However, while Seo et al. (2022) [7] with its Stackelberg game-based algorithm provided only theoretical verification, Chen et al. (2022) [11] verification evaluated its Greedy algorithm using real-world experiments. Those suggesting the need to apply real-world experiments or at least close to real-world simulation to evaluate the solution more carefully. Also, both solutions only solved for singular subjects, making their solutions less heterogeneous than they possibly could.

Among the most recent solutions tackling only latency in the MEC environment, [16], [6] proposed more complex RL solutions than their predecessors. In particular, Liu et al. (2023) [16] applied deterministic policy gradient (DDPG), while Zeng et al. (2024) [6] applied Multi-agent DRL. Both those works suggest that DRL-based solutions are quite efficient solving for such complex environments as MEC with Multiple subjects. Both of their works emphasize the importance of considering heterogeneous environments, however, as their predecessors, they only provided a simulation without considering Vehicles as their subjects.

2.1.2 Energy as the Optimization Target

Another route for the research in MEC is the optimization of energy efficiency. Despite being less popular than Latency optimization due to its challenges in modelling the energy expenditure close to the real world, there are still several works applying various solutions. For instance, both in [15] and [18], authors concentrated on solving for energy optimization in the MEC environment. While in [15], authors employed a more general technique with the difference-of-convex algorithm (DCA). An extension of the previous work is presented in [18], where authors lead their solution further by applying DRL with sequence-to-sequence (S2S) neural networks suggesting RL-based solutions are superior in terms of solutions in MEC environments. However, none of these solutions applied practical experimentation for verification of their solutions unlike the ϵ-bounded approximation algorithm proposed in [13] making its work more applicable to real-world situations. However, the downside of this work is solving only

for singular subjects those in a more simplistic environment than its counterparts.

Further, a more sophisticated technique applying the DDPG model was introduced in [8], where authors focused on DNN inference tasks across MEC. Similarly, to Latency Optimization, these works did not focus on Vehicles as their subject of UE description leaving complexities introduced by Vehicles with their mobility. On the other hand, in [20] authors addressed the problem of the energy efficiency of vehicles as a subject by investigating Wireless Energy Harvesting. However, while addressing vehicles this work did not consider latency optimization which is critical for such real-time environments as VEC.

2.1.3 Latency and Energy as the Optimization Target

An increasing amount of studies have started to pay attention to the trade-off between latency and energy consumption during task offloading. One of the earlier solutions that introduced DRL in combination with S2S neural networks was proposed in [21]. Their goal was to optimize energy and latency for IoT devices. In a similar vein, [9], and [14] balanced both measures in General User Equipment using Double-DQN and DRL with S2S neural networks, respectively. However, their frameworks did not consider the mobility of Vehicular subjects, and their results do not apply to such environments as in many previous works.

An existing offloading framework called The Nimbus was proposed in [19] that considers both energy and latency optimisation but in the environment of Augmented Reality (AR) applications. Although this framework works in a complex environment of AR which is a step forward compared to more General User Equipment, they still have a lack of mobility that could be introduced with complexities of Vehicular environments. The problem of balancing latency and energy was reexamined by Lin et al. (2020) [5] for the DNN Inference task as the new subject for dual optimization. Even though DNN inference tasks require very complex environments and conditions they still lack vehicular mobility.

The analysis of previous research reveals a substantial gap in the present literature: just a handful of solutions in MEC address the issue of both latency and energy

consumption optimization, among which there is a noticeable lack of concern for the Vehicular environment. Although some research has concentrated on one of these elements (latency or energy), few have optimized both. Furthermore, because these solutions rely heavily on simple simulations it is still unclear how they will behave in close to real-world scenarios.

2.2 Development Environment

2.2.1 Software section

This part introduces software stack used for both development and validation of the Proposed work in VEC environment. The software implementation including improvements made by previous research can be found in my Github repository [22].

RAPID framework

The core of the real world-simulation is a RAPID offloading framework. RAPID is and existing open-source offloading framework designed to offload task using virtual machines and leveraging JAVA. The framework was modified by our previous research projects including [1] and [2]. This includes such updates to an original RAPID implementation as:

- Addition of Decision Engine that allowed a mechanism to utilize Machine Learning models by creating an interface for actual RAPID scheduler to utilize.

- Integration of an energy aware node activity manager that utilizes built-on Jetson Nano Wake-on-LAN feature.

- Implementation of a request Queue that keeps track of task that were rejected by edge cluster due to overloading of edge nodes and records Turnaround time (time spent in queue).

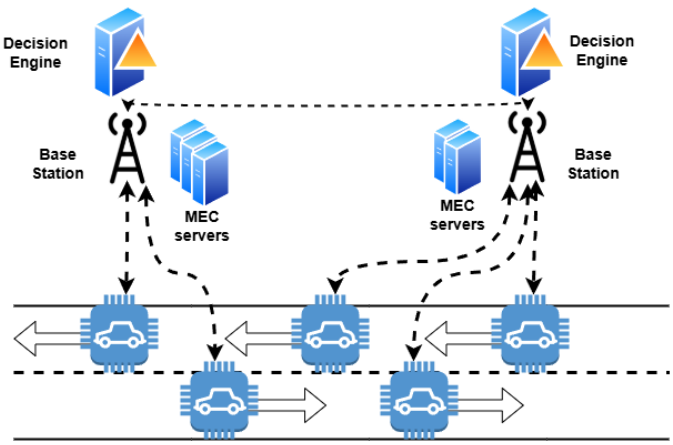

Figure 2-1: Architecture of VEC workflow.

Simulation of Mobility

We model our VEC environment as a hierarchical system where mobile vehicles (MVs) generate computational tasks that can be processed by MEC clusters (MCs) deployed at base stations (BSs). The workflow and the relations between parts of the MEC framework are illustrated in Figure 2-1. Specifically, the workflow depicts the connection between vehicles to the closest BSs from where they could access the MCs resources. Also, there is a connection between a base station in the case that to mobility decision engine decides to offload the function to the next base station. Each base station has its own Decision Engine. Noting that the delay between the decision engine and the Base station could be neglected because of the manageable size of the load transfer and proximity. This architecture is specifically designed to enable these adaptive strategies through reinforcement learning. The system architecture is characterized by:

- Base Station $\mathcal{B} = \{1, 2, ..., M\}$: fixed access points that provide a connection between vehicles and edge computing resources. For the simulation, 10 base stations were chosen and placed on the Astana city topology, which is depicted in Figure 2-2.

- The cluster of edge nodes $\mathcal{N}_b = \{1, 2, ..., N_b\}$: the main computational unit of the system, is connected to the base station $b \in \mathcal{B}$, with each hosting 20 edge

nodes for the simulation environment.

- Wake-on-LAN (WoL) nodes $\mathcal{A}_b \subset \mathcal{N}_b$: a subset of active nodes that are idle most of the time to conserve energy but can be activated on demand by the Decision Engine.

- Mobile vehicles $v \in \mathcal{V}$: a source of offloading tasks that travel across the road network with mobility patterns simulated using SUMO.

- Decision Engine: a scheduler component of the system that is responsible for making offloading decisions. It determines whether to offload a specific edge node or to wake up a WoL node to handle the workload.

Each of the above components is designed to reflect real-world constraints both in terms of realistic signal propagation due to mobility in urban environments, and energy costs of the computation edge nodes. OpenStreetMaps[23] For the simulation

Figure 2-2: SUMO map config for Astana city.

setup of vehicular mobility, a real-world like road network image of Astana city was utilized. The topological data was extracted from OpenStreetMap via the bbbike.org extraction service [23] in standard .osm format. The road network was generated using SUMO's netconvert application. The output is visually present in Figure 2-2.

To initiate the mobility of vehicles across this road network, SUMO's randomTrips.py application was utilized with a seed of 42, Poisson distribution with lambda 1.5 and a duration of 3600 seconds which is a sufficient mobility pattern.

2.2.2 Hardware section

Jetson Nano Cluster

For a validation in a real-world testbed, the cluster of Jetson Nano devices was utilized. Such Jetson Nano kits are suitable as edge nodes because of their relatively small power usage (5W) and high compuattion capabilites:

- 20 NVIDIA Jetson Nano Developer Kits (10 with Wake-On-Lan capabilities, 10 without)
  - Quad-core ARM A57 CPU @ 1.43 GHz
  - 4GB LPDDR4 RAM
  - 128-core NVIDIA Maxwell GPU
  - 16GB eMMC storage

This comprehensive hardware setup allowed us to accurately measure energy consumption patterns, validate the performance of our offloading strategies in realistic conditions, and collect empirical data that informed our simulation environment.

Chapter 3

Methodology

This section discusses the methodology for our RL-based task offloading algorithm and its VEC environment components. It consists of mobility simulation using SUMO, placement of base stations using K-means algorithm, a latency model, an energy consumption model based on real-world measurements, and two RL-based offloading models: Feed-Forward DQN (FF-DDQN) and an advanced Transformer-Enhanced DDQN (TE-DDQN).

3.0.1 Energy-Aware Vehicular Edge Computing System

3.0.2 Base Station Placement Using K-means Clustering

As for the placement of base stations to strategically cover the whole map, a K-means clustering algorithm was utilized. The algorithms ensure that the placement is optimized to cover the entire map and provide comprehensive coverage encompassing all routes of MVs. Figure 3-1 illustrates the base station placement of 10 base stations on the Astana road network.

Algorithm 1 outlines the K-means clustering procedure for base station placement. It inputs the road network nodes extracted from the generated Astana city map, and the number of base stations. After randomly generating base stations' placements, it iteratively updates the centroids' positions based on the Euclidean distance of the nearest road nodes. The result is the placement of base stations that cover

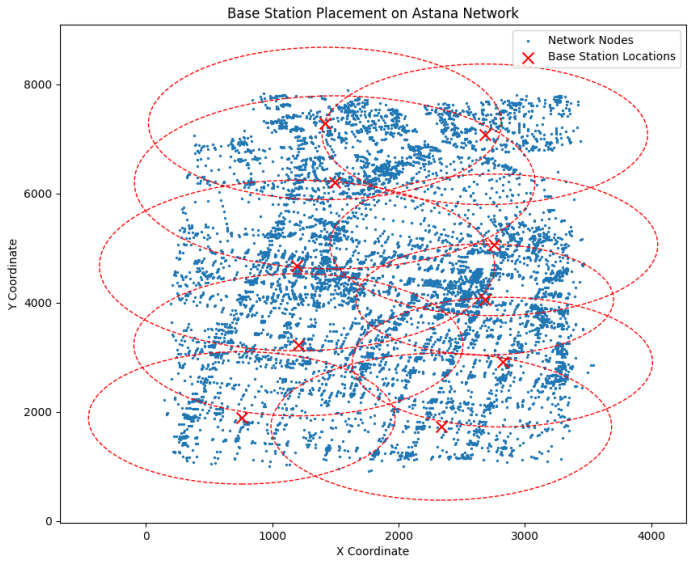

Figure 3-1: Base Station Locations.

all road nodes and minimize average distance from potential vehicles to the base stations. The approach in itself in quite adaptable because of easy modifications for different numbers of base stations and various topologies of road maps. Also, it avoids unnecessary complexities like other solutions might bring, such as genetic algorithms in terms of both computational and implementation complexity.

The connectivity between vehicles and base stations is modelled using a distance-based approach, where the signal strength and data rate decrease as the distance increases, affecting the transmission latency for task offloading.

3.0.3 Real-World Energy Consumption Model

Task Generation Approach

Instead of depending on a purely simulated model for the energy consumption for the VEC environment, an experiment to gather empirical data was conducted. The RAPID offloading framework [1] can offload real-world-like path-finding computation to an edge node. The NVIDIA Jetson Nano device was chosen as the realistic representation of an edge node due to its energy efficiency (5W consumption) and

Algorithm 1 K-means Clustering for Base Station Placement

Input: Road network nodes $\mathcal{N} = n_1, n_2, ..., n_m$ with coordinates, number of base stations k
Output: Base station locations $\mathcal{B} = b_1, b_2, ..., b_k$ and coverage radii $\mathcal{R} = r_1, r_2, ..., r_k$
Initialize centroids $\mu_1, \mu_2, ..., \mu_k$ randomly from $\mathcal{N}$
converged $\leftarrow$ false
while not converged **do**
    for each node $n_i \in \mathcal{N}$ **do**
        cluster$(n_i) \leftarrow \arg\min_j d(n_i, \mu_j)$ where d is the Euclidean distance
    for $j \leftarrow 1$ to k **do**
        $\mu_j \leftarrow \frac{1}{|\mathcal{C}_j|} \sum_{n_i \in \mathcal{C}_j} n_i$ where $\mathcal{C}_j$ is the set of nodes in cluster j
    if centroids no longer change significantly **then**
        converged $\leftarrow$ true
$\mathcal{B} \leftarrow \mu_1, \mu_2, ..., \mu_k$
$\mathcal{R} \leftarrow r_1, r_2, ..., r_k$
return $\mathcal{B}, \mathcal{R}$

computational capabilities. [2]

The first step in the proposed method was to create a real-world workload for the system, which included data and computational tasks that could be offloaded. The subject for this task was the Rosbot Plus mentioned above. This robot used the data generated from the attached sensors and processed data based on ROS1 Melodic to navigate from an arbitrary point A to point B. To build a path, the robot processed all the gathered data into a CostMap (65KB). It employed the Dijkstra algorithm, which has significant time complexity due to the nature of graph traversal computations. Utilizing a priority queue had a time complexity of $O((V + E) \log V)$ where E nrepresented the number of edges and V the number of vertices. However, the algorithm's space complexity remained manageable $O(V)$. [CITATION NEEDED!]

To create a variability of the load on the edge node, it was decided to implement 10 different task offloading scenarios. Each scenario represents a common workload and requirements from a Rosbot Plus in a VEC environment. Each of the scenarios is modelled as a Directed Acyclic Graph to diverge the load on the edge node with a clear structure and a potential to allow more granular offloading decisions, such as deciding on each small task within complex computation. Each of the 10 scenarios

tests the behaviour of a Jetson Nano under various workloads:

1. Linear Chain Scenario: strict sequential execution of 15 consecutive tasks representing a non-parallelizable workload.

2. Fork and Merge Scenario: execution of two parallel tasks followed by another two, and merged into one task. This represents the behaviour of edge nodes under parallel computation concerning synchronization.

3. Parallel Tasks Scenario: parallel workload consists of 5 parallel tasks to test the edge node under parallelizable computations.

4. Sequential with Branching Scenario: sequential task followed by parallel and back tests of the device on a combined workload.

5. Complex Merge Scenario: This pattern consists of 4 parallel tasks followed by 4 sequential merge tasks, which depict highly aggregated tasks.

6. Double Fork Scenario: two initial parallel processes, where each fork into two other tasks, followed by a large task. It tests the ability to handle nested parallel tasks.

7. Loop Replanning Scenario: Two initial parallel tasks that each fork into two processes. Afterwards, a planning task merges the evaluation results. This is followed by three parallel execution tasks of varying workloads. It tests the ability to handle dynamic feedback in iterative computational processes.

8. High Parallelism Scenario: massive parallel workload consisting of 30 parallel tasks to test the edge node under highly parallelizable computation with an increased workload in comparison to the Parallel Tasks Scenario.

9. Mixed Workload Scenario: Four initial sequential tasks followed by six parallel tasks in the second phase, with two parallel merge tasks for final consolidation. It tests the agent's ability to transition smoothly between sequential and parallel execution phases.

10. Extended DAG Scenario: This scenario evaluates the agent's capability to optimize for long critical paths while effectively utilizing the limited parallelism in the middle stage

Task Parameters

Each of the tasks τ mentioned above task DAGs are characterized by the following parameters:

- Data Size d_τ: The amount of data that must be transferred for task execution, ranging from 375KB (for 5-task scenarios) to 2,000KB (for 30-task scenarios).
- Required CPU Cycles λ_τ: Computational resources needed, ranging from 300 to 1,123 megacycles, measured using `perf stat` tool during the empirical study.
- **Deadline**ϵ_τ: Task completion time constraint generated using normal distribution varying rom 6 to 10 seconds reflecting the latency sensitivity of path-finding applications.
- Scenario ID s_τ: Integer (1-10) identifying the task scenario type for energy model mapping. Helps to link empirical measurements to the task workload during simulation.

Those parameters were collected from empirical measurements. Scenarios were offloaded to the Jetson Nano, and comprehensive metrics were collected during execution using Docker stats and Tegrastats, including:

- Energy consumption through power measurements by the sensor
- CPU utilization across multiple cores and temperature readings
- Task execution time from start to completion

Each scenario represents a distinct computational pattern with different implications for task scheduling, resource allocation, and energy consumption, resulting in a more heterogeneous environment.

Empirical Energy Modeling

From this rich dataset, we derived an energy model that maps each task scenario to its corresponding energy consumption profile:

$$E_{\text{proc}} = P(s_\tau) \times d \times (\alpha(s) \times l) + \eta \tag{3.1}$$

Where:

- E_{proc}: Energy consumption (Joules)
- s_τ: Task scenario ID (1-10)
- d: Task duration (seconds)
- l: Node load (0-1)
- $\alpha(s_\tau)$: Load impact factor for scenario s, calculated as: $\max(0.05, |\text{corr}(\text{CPU\_util}, \text{power})|)$
- $P(s_\tau)$: Base power consumption for scenario s (Watts), obtained from empirical measurements
- η: Gaussian noise based on measured variance: $\mathcal{N}(0, \sqrt{\text{variance}} \times 0.1)$

Each task scenario has a unique base power consumption $P(s)$ derived from empirical measurements, unique for each task scenario ID. To model the non-linear relation between system load and power consumption $\alpha(s_\tau)$ is introduced which is an empirical measured correlation between CPU utilization and power consumption. Real-world energy consumption exhibits stochastic variations due to thermal effects, background processes, and measurement noise. In this formula, it was modelled using Gaussian noise.

Wake-up Energy Function

To preserve the energy in a low-workload period, the system can put some WoL-capable nodes to sleep. However, to make the system closer to a real-world the transition between sleep and active state must be included:

$$E_{\text{wake}} = E_{\text{base}} \times \phi(t) \tag{3.2}$$

Where:

- E_{wake}: Energy required to wake up a node (Joules)
- E_{base}: Base wake-up energy (5.0 Joules)
- t: Node type (standard, high_performance, low_power)
- $\phi(t)$: Type factor (2.0 for high_performance, 0.5 for low_power, 1.0 for standard)

Edge nodes can differ in their operational costs, the wake-up energy consumption would also be factored $\phi(t)$ ensuring this diversity, making the environment more heterogeneous. An arbitrary placeholder of 5 Joules was used E_{base} , which in the future can be empirically measured to reflect the actual consumption more accurately.

Idle Power Consumption

Even without active processing, edge nodes maintain steady power consumption. Based on empirical measurements, the model of the idle power consumption at the edge node is:

$$E_{\text{idle}} = P_{\text{base}} \times \phi(t) \times d_i \tag{3.3}$$

Where:

- E_{idle}: Idle power consumption (Joules)
- P_{base}: Base idle power (empirically measured 2.0 Watts for Jetson Nano)
- t: Node type (standard, high_performance, low_power)
- d_i: time taken in idle state
- $\phi(t)$: Type factor (2.0 for high_performance, 0.5 for low_power, 1.0 for standard)

Total Consumed Energy

The total energy consumption in our VEC environment encompasses all energy expenditures across the system components. To accurately reflect real-world energy costs, we model the total consumed energy as the sum of three primary components:

$$E_{consumed} = E_{\text{proc}} + E_{\text{wake}} + E_{\text{idle}} \tag{3.4}$$

3.0.4 Simulated Latency Model

Alongside energy efficiency, task completion latency is a critical performance metric in VEC environments. Many vehicular applications have strict timing requirements, and excessive latency can render services unusable or even dangerous in safety-critical scenarios. We develop a comprehensive latency model that captures the key components of end-to-end task completion time, including wireless transmission, processing, and queueing delays.

The transmission latency for a task from vehicle v to base station b is modeled as:

$$\delta_{trans}(v, b) = \frac{d_v}{r(v, b)} + L_{base} \tag{3.5}$$

Where:

- δ_{trans}: Transmission latency (seconds)
- d_v: Task data size (bits)
- $r(v, b)$: Achievable data rate (bits per second)
- L_{base}: Base latency due to protocol overhead (typically 5-10ms)

The achievable data rate $r(v, b)$ is calculated based on the Shannon capacity formula, taking into account the Signal-to-Interference-plus-Noise Ratio (SINR):

$$r(v, b) = W \times \min(8.0, 0.75 \times \log_2(1 + \text{SINR}(d, i))) \tag{3.6}$$

Where:

- W: Bandwidth (Hz), default 20MHz, reflecting typical channel bandwidth in urban LTE deployments

- 8.0: Maximum spectral efficiency cap (bits/symbol) based on highest-order modulation schemes in LTE

- 0.75: Practical efficiency factor accounting for protocol overhead, guard intervals, and implementation losses

- $\text{SINR}(d, i)$: Signal-to-Interference-plus-Noise Ratio, which depends on distance d and interference i

This data rate calculation captures several important real-world effects, like the spectral efficiency cap and efficiency factor account for practical limitations in wireless systems that prevent achieving the theoretical Shannon capacity.

The SINR is a key determinant of achievable data rates in wireless systems:

$$\text{SINR}_{\text{dB}}(d, i) = P_{\text{rx\_dBm}}(d) - 10 \times \log_{10}(N_{\text{mW}} + \sum P_{\text{int\_mW}}(i)) \tag{3.7}$$

Where:

- SINR_{dB}: SINR in decibels

- $P_{\text{rx\_dBm}}$: Received power (dBm) = $P_{\text{tx\_dBm}} - \text{PL}(d)$

- $P_{\text{tx\_dBm}}$: Transmit power (23 dBm for base stations)

- N_{mW}: Noise floor in milliwatts (converted from -95 dBm, typical for urban environments)

- $P_{\text{int\_mW}}$: Interference power from other base stations in milliwatts

This SINR calculation incorporates both thermal noise and interference from other base stations, providing a more realistic model than approaches that consider only distance-based path loss:

$$\mathrm{PL}(d) = 20 \times \log_{10}\left(\frac{4\pi \times f \times 10^9}{3 \times 10^8}\right) + 10 \times \beta \times \log_{10}(\max(1.0, d)) \tag{3.8}$$

Where:

- PL: Path loss (dB)
- f: Carrier frequency (GHz), default 2.4GHz chosen to represent typical unlicensed band deployments
- β: Path loss exponent (3.5 arbitrary number)
- d: Distance (meters)

This model combines free-space path loss (first term) with environment-specific attenuation (second term) to provide a realistic estimate of signal strength at different distances. The path loss exponent $\beta = 3.5$ reflects typical urban environments with a mix of line-of-sight and non-line-of-sight paths.

Processing Latency

The time taken for the edge node to computate the task is modelled as:

$$\delta_{proc}(v, n) = \frac{\lambda_v}{C_n} \times load_n \tag{3.9}$$

Where:

- δ_{proc}: Processing latency (seconds)
- λ_v: Required CPU cycles (megacycles)
- C_n: Computational capacity of node n (megacycles/second)
- $load_n$: Current load of the node (0-1)

This model captures the performance degradation that occurs when multiple tasks compete for resources on the same node. As node load increases, processing time increases proportionally, reflecting resource contention, context switching overhead, and cache interference. The node capacity C_n is set to 500 megacycles per second for standard nodes, representing the effective computational throughput of a Jetson Nano under typical operating conditions.

Total Latency

The total latency for a task:

$$\delta_{total} = \delta_{trans\_up} + \delta_{proc} + \delta_{trans\_down} + \delta_{queue} \tag{3.10}$$

Where:

- δ_{trans_up}: Uplink transmission latency (from vehicle to base station)
- δ_{proc}: Processing latency on the edge node
- δ_{trans_down}: Downlink transmission latency (from base station to vehicle)
- δ_{queue}: Queueing delay (if the task is queued)

This comprehensive model captures all significant contributors to end-to-end latency in a VEC environment. The environment also includes the possibility of the task awaiting the resources in a queue.

3.0.5 Problem Formulation

We formulate the task offloading problem as a Markov Decision Process (MDP) with the following components:

State Space

The state s_t is a representation of the system's condition at time t:

- task size: Data size d_τ in KB representing the workload of the task

- required CPU cycles: required CPU cycles in megacycles by the task
- task deadline: Deadline in seconds
- task scenario id: Scenario ID s_τ (1-10), which connects offloaded task to empirically measured variables
- vehicle position: Position coordinates (x_v, y_v) of the vehicle
- vehicle speed: Speed v_v in m/s, indicating vehicle mobility
- distance to bs: Distance to nearest BS in meters
- queue ength: Queue length of current each Base station
- active nodes: Number of active nodes representing available computational resources
- node active_status: a boolean value showing which node is in the awake state
- energy efficiency factor: Energy efficiency ratio for the current task scenario

When flattened, this state representation contains approximately 200 features.

Action Space:

For each task, the action a_t is discrete and can be:

- Offload to the edge node n of the cluster, where $n \in \{0, 1, ..., N_b - 1\}$
- Wake up an idle node, represented as an action N_b

With 20 edge nodes per base station, the action space size is 21, with 21 representing wake action.

Reward Function

The reward function is carefully designed to balance latency optimization and energy efficiency:

$$R(s_t, a_t) = \begin{cases} (1 - w_e) \times \left(1 - \frac{\delta_{total}}{\epsilon_v}\right) + w_e \times (1 - E_{consumed}), & \text{if task completed} \\ -0.2, & \text{if task queued} \\ -1.0, & \text{if task rejected} \end{cases} \tag{3.11}$$

where w_e the energy weight factor balances the trade-off between latency and energy, δ_{total} the total processing time ϵ_v is the task deadline, $E_{consumed}$ is the energy consumed for the given scenario and duration (from our energy model), and $E_{max} =$ 100 Joules is a normalization factor for maximum possible energy consumption.

This reward structure incentivizes:

- Fast task completion (higher reward for lower δ_{total})
- Energy efficiency (higher reward for lower $E_{consumed}$)
- High task completion rate (the penalty for rejected tasks)

3.0.6 Solution Implementation

Feed-Forward DQN (FF-DDQN)

Our first approach implements a standard RL model called DQN for the task offloading problem. DQN is a well-suited and popular Reinforcement Learning solution for environments with discrete action spaces. The DQN architecture is suitable for processing high-state dimensional spaces, such as task offloading in a VEC environment. Even though the DQN approach is potentially limited in its ability to leverage temporal patterns in the historical data, it can serve as a strong baseline for task-offloading decision learning.

Network Architecture: The DQN network consists of:

- Input layer matching the state features in a flattened vector
- Three hidden layers: 256 neurons each with ReLU activation to capture the complex relationship between features
- Dropout (0.1) applied after each hidden layer to avoid overfitting by adding noise
- Output layer: 21 neurons dimension to match action space
- Orthogonal weight initialization for better convergence

Training Algorithm: The DQN agent learns an action-value function $Q(s, a; \theta)$ approximated by the neural network with Q weights θ, state space s_t and action space a_t. The training process follows the standard DQN algorithm:

Exploration Strategy: For exploration, we implement an ϵ-greedy strategy with an initial value of $\epsilon = 1.0$, decaying by a factor of 0.995 after each episode with a minimal value of $\epsilon = 0.05$. This standard approach helps to leverage the first period of exploration and then smoothly transition to the exploitation of the replay buffer of DQN.

Key details to the standard FF-DQN algorithm include:

- Gradient clipping to prevent exploding gradients
- Smooth L1 loss (Huber loss) for robustness to outliers
- Periodic saving of best models based on evaluation performance

Transformer-Enhanced DDQN (TE-DDQN)

Our second approach extends the DQN framework with a Transformer architecture to better capture temporal patterns in the sequential data such as modelled MDP. The key motivation is to leverage historical load information to predict future congestion based on historical patterns. Unlike simple FF-DQN, the Transformer model can

Algorithm 2 FF-DQN Training for Task Offloading

Initialize replay buffer $\mathcal{D}$ with capacity $N_{episodes}$
Initialize action-value network Q with random weights θ
for episode = 1 to $N_{episodes}$ **do**
    Observe initial state s_1
    for $t = 1$ to T_{step} **do**
        With probability ϵ, select random action a_t
        Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$
        Execute action a_t, observe reward r_t, next state s_{t+1}, and done signal d_t
        Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
        Sample mini-batch of transitions $(s_j, a_j, r_j, s_{j+1}, d_j)$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j, & \text{if } d_j = \text{True} \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'; \theta'), & \text{otherwise} \end{cases}$
        Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to θ
        Update exploration rate: $\epsilon \leftarrow \max(\epsilon_{min}, \epsilon \times \epsilon_{decay})$
return Chosen action a_t

identify trends and patterns across several time stamps. This is particularly valuable in a VEC environment where load patterns may initiate similar patterns across wide processes.

Network Architecture: The Transformer-based agent architecture consists of:

- State normalization layer to standardize input features
- Input embedding layer that projects each state vector to a 128-dimensional space
- Positional encoding to incorporate sequence order information:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}), \quad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}) \tag{3.12}$$

- Transformer encoder with 3 layers, each containing:
  - Multi-head self-attention mechanism (4 heads) with scaled softmax attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3.13}$$

  - Position-wise feed-forward network: $\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$

  - Layer normalization and residual connections
  - Dropout (0.1) to avoid overfitting (same as DQN)

- Dueling network architecture that separates state value and advantage functions:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a') \right) \tag{3.14}$$

  where:

  - Value stream: 3-layer MLP with layer normalization and ReLU activation
  - Advantage stream: 3-layer MLP with layer normalization and ReLU activation

Training Algorithm: The Transformer-based approach uses a modified DDQN algorithm that works with sequences of states. And have same exploration/exploitation strategy as FF-DQN:

Key differences from the standard DQN approach include:

- Sequential replay buffer $\mathcal{D}$ that stores sequences of transitions
- State history buffer that maintains the recent state sequence for decision-making
- Soft target network updates with parameter $\tau = 0.01$

Training and Evaluation Framework

FF-DQN and TE-DDQN agents are trained within a comprehensive framework that ensures fair evaluation. The training process for each agent includes:

Training Parameters:

- Discount factor $\gamma = 0.99$
- DQN learning rate $\alpha_{DQN} = 0.001$
- Transformer learning rate $\alpha_{Transformer} = 0.0003$
- Batch size: 64

Algorithm 3 Transformer-Based DQN Training for Task Offloading

Initialize sequential replay buffer $\mathcal{D}$ with capacity $N_{episodes}$
Initialize Transformer policy network Q with random weights θ
Initialize target network Q' with weights $\theta' \leftarrow \theta$
Initialize state history buffer $\mathcal{H}$ with capacity H
Initialize replay memory $\mathcal{M}$ with capacity M
for episode = 1 to $N_{episodes}$ **do**
    Observe initial state s_1
    Clear state history buffer $\mathcal{H}$
    for $t = 1$ to T_{step} **do**
        Add s_t to state history buffer $\mathcal{H}$
        With probability ϵ, select random action a_t
        Otherwise, select $a_t = \arg\max_a Q(\mathcal{H}, a; \theta)$
        Execute action a_t, observe reward r_t, next state s_{t+1}, and done signal d_t
        Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in episode buffer
        If episode buffer has at least H transitions, add sequence to $\mathcal{D}$
        Sample mini-batch of sequences from $\mathcal{D}$
        For each sequence, compute target values using target network Q'
        Perform gradient descent step on loss for θ
        Update exploration rate: $\epsilon \leftarrow \max(\epsilon_{min}, \epsilon \times \epsilon_{decay})$
    If episode buffer has remaining transitions, add final sequences to $\mathcal{D}$
    if episode mod soft_update_freq = 0 **then**
        Soft update target network: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
    Update learning rate using a scheduler

- DQN replay buffer capacity: 100,000 transitions
- Transformer sequential replay buffer capacity: 100,000 sequences
- DQN target network update frequency: 10 episodes
- Transformer soft update parameter: $\tau = 0.01$
- Training episodes: 1,000
- Maximum steps per episode: 600
- Random seed: 42 (for reproducibility)

Evaluation Metrics: The performance of both approaches is evaluated using a comprehensive set of metrics:

- **Average Reward:** Mean reward per episode, reflecting overall performance
- **Task Completion Rate:** Percentage of tasks successfully processed within their deadline
- **Task Rejection Rate:** Percentage of tasks rejected due to resource constraints
- **Task Drop Rate:** Percentage of tasks dropped due to deadline expiration
- **Average Latency:** Mean processing time from task generation to completion
- **Energy Consumption:** Average energy usage in Joules per task
- **Energy-Task Ratio:** Energy consumption divided by task completion rate, indicating energy efficiency
- **Node Utilization:** Average percentage of active nodes across base stations
- **Convergence Speed:** Number of episodes required to reach 95% of maximum reward
- **Reward Variance:** Stability of performance across episodes

Evaluation Process: Both models are evaluated at regular intervals during training (every 50 episodes) and after training completion. The evaluation process consists of:

- Running 10 evaluation episodes with exploration disabled ($\epsilon = 0$)
- Computing the average and standard deviation for all metrics across these episodes
- Comparing performance between the two approaches using statistical significance tests
- Generating visualizations to illustrate performance differences and training progression

At the end of training, both models undergo a final comprehensive evaluation to determine their ultimate performance. This includes extended evaluation runs to minimize variance and provide high-confidence results.

Chapter 4

Results

4.1 Training Performance Analysis

4.1.1 FF DQN training results

The DQN approach demonstrated robust performance in the task offloading scenario with the following results:

- Achieved an average reward of 220.6 after 1,000 training episodes
- Reached a task completion rate of 99.8%
- Maintained an average latency of 1.9 seconds per completed task
- Consumed an average of 19 Joules of energy per task
- Exhibited convergence to near-optimal performance after approximately 516 episodes

The training results are also depicted in Figure 4-1 , where DQN refers to FF-DQN and Transformer refers to TE-DDQN approaches.

The DQN agent successfully learned to balance latency minimization and energy efficiency, as evidenced by its node selection patterns. Analysis of the agent's behaviour revealed a preference for offloading tasks to nodes with lower current loads, indicating effective load balancing.

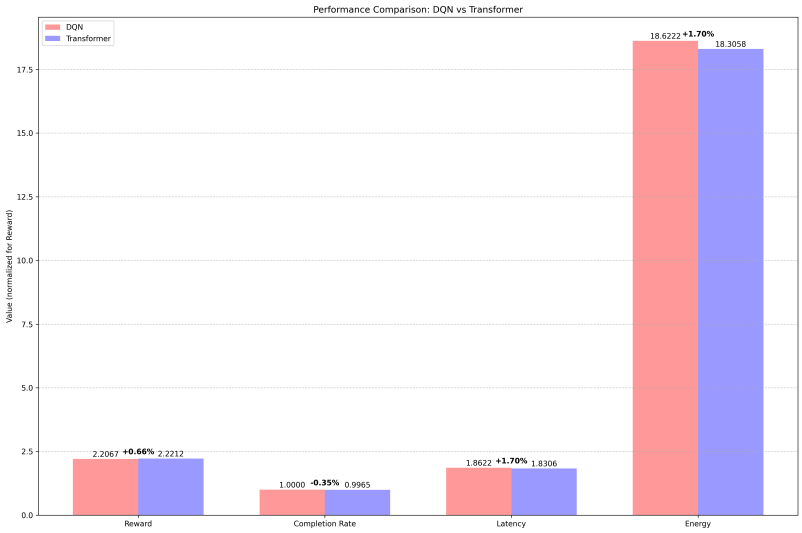

Figure 4-1: performance comparison

Interestingly, the DQN agent developed a strategy of maintaining a consistent number of active nodes (around 13-14 per base station) rather than frequent activation and deactivation. This suggests that the agent learned the energy penalty associated with wake-up operations and adapted its policy accordingly.

The exploration rate decay from 1.0 to 0.05 over the training period allowed the agent to transition from random exploration to exploitation of its learned policy. This transition is visible in the reward curve, which shows increasing stability in later episodes.

4.1.2 TE DDQN training results

The Transformer-based approach demonstrated several advantages over the standard DQN, particularly in its ability to leverage temporal patterns in the historical data:

- Achieved an average reward of 222.1 after 1,000 training episodes (0.7% improvement over FF-DQN)
- Reached a task completion rate of 99.65%
- Maintained an average latency of 1.83 seconds per completed task (3.6% reduction)

- Consumed an average of 18.3 Joules of energy per task (3.7% reduction)
- Exhibited faster convergence, reaching near-optimal performance after approximately 8 episodes (98.45% faster)

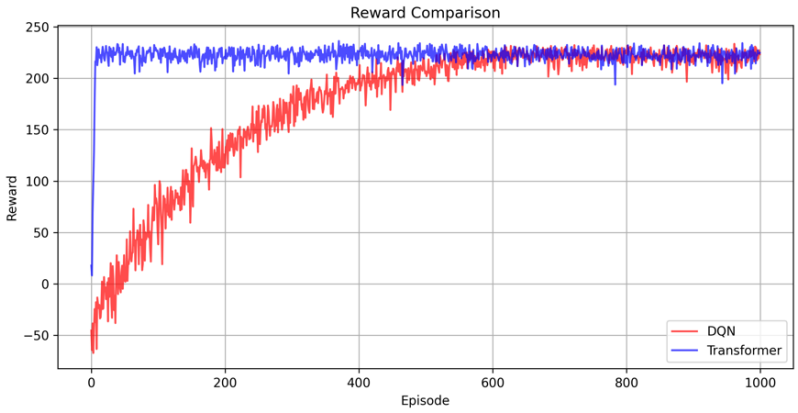

Figure 4-2: Training Reward Convergence

The Transformer model's attention mechanism allowed it to focus on relevant historical states when making decisions. Analysis of the attention weights revealed interesting patterns. The sequential nature of the Transformer's input allowed it to better anticipate future congestion based on historical patterns. This led to more efficient node management, with lower energy consumption and higher task completion rates compared to the DQN approach.

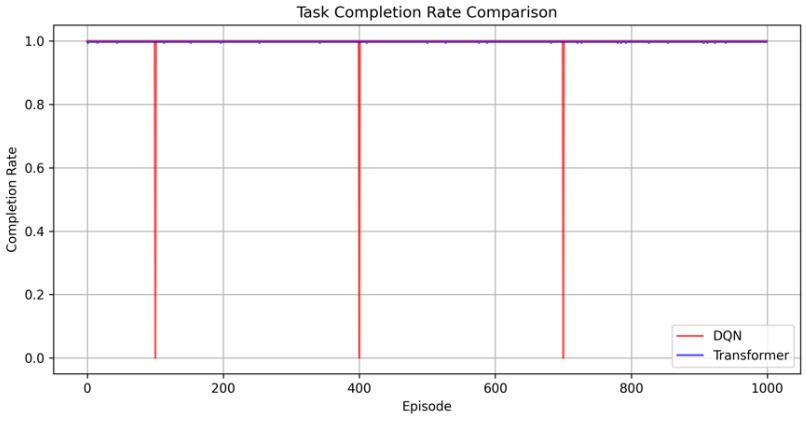

Figure 4-3: Training Trask Completion Rate

Figure 4-2 illustrates the TE-DDQN training progression, showing faster convergence and more stable performance compared to the FF-DQN approach. The reward

variance in the whole training period shows a much more stable process than those presented by FF-DQN.

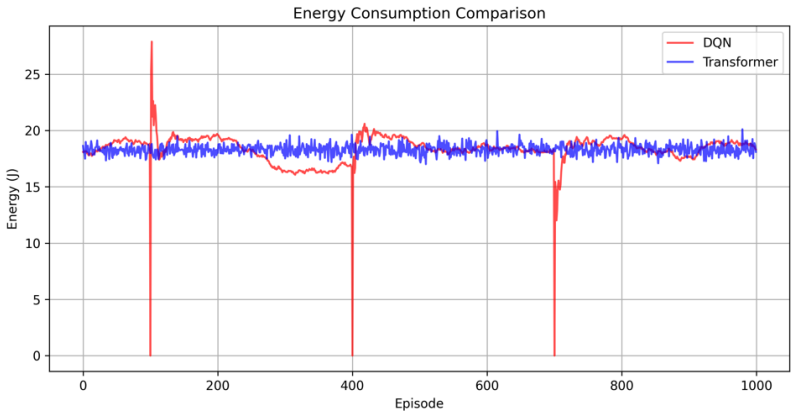

Figure 4-4: Training Energy Consumption

Figure 4-3 depicts almost the same completion rate for both approaches, resulting in near-perfect task completion rates. However, it is also evident from the Figure that occasional performance degradation of FF-DQN about every 300 steps during both exploration (episodes 100 and 400) and exploitation (near episode 700) periods. This again suggests the formidable stability in training of the transformer-based approach.

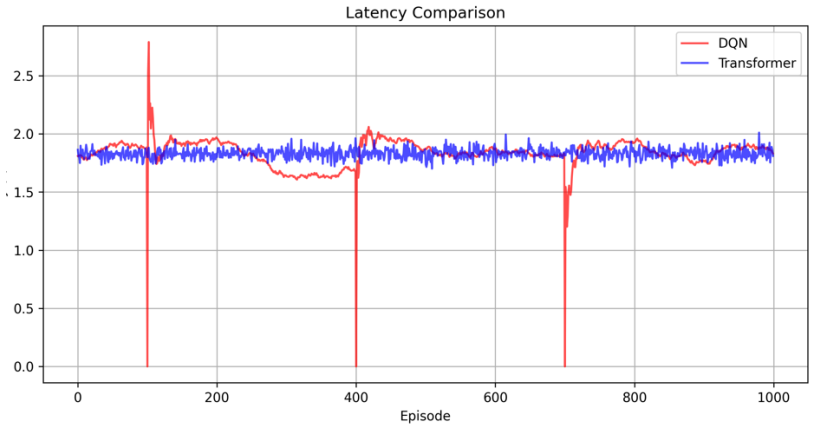

Figure 4-5: Training Latency

Both Figure 4-4 and 4-5 demonstrate sudden spikes and much less stable training from FF-DQN compared to TE-DDQN's energy and latency results. Being the main optimization target, TE-DDQN establishes a much more suitable variation of RL-

Model for training and performance in the VEC environment where such sudden drops in performance are not desired.

4.2 Comparative Analysis

4.2.1 Hyperparameters and Training Configuration

For a fair comparison between approaches, we carefully standardized the training configuration where appropriate while optimizing hyperparameters separately for each architecture. Common parameters include:

- **Discount factor** $\gamma = 0.99$: Standard value for episodic tasks, balancing immediate and future rewards.
- **Exploration parameters**: Both approaches use ϵ-greedy exploration with initial $\epsilon = 1.0$, minimum $\epsilon = 0.05$, and decay factor 0.995. This schedule provides sufficient exploration while transitioning to exploitation at a reasonable rate.
- **Replay buffer capacity**: 100,000 transitions/sequences for both approaches, providing sufficient diversity in training examples.
- **Batch size**: 64 for both approaches, a standard value that balances computational efficiency and statistical stability of gradient estimates.
- **Training episodes**: 1,000 for both approaches, providing sufficient opportunity to learn effective policies.
- **Maximum steps per episode**: 500, matching the realistic duration of vehicle journeys in our simulation.
- **Random seed**: 42 for reproducibility, with additional seeds used for validation runs to assess robustness.

Architecture-specific hyperparameters were tuned separately to ensure optimal performance for each approach:

- **FF-DQN**:
  - Learning rate: 0.001
  - Target update frequency: 10 episodes
  - Hidden layer sizes: [256, 256, 128]
  - Dropout rate: 0.2
- **TE-DDQN**:
  - Learning rate: 0.0003 with StepLR scheduling (halving every 400 episodes)
  - Soft update parameter: $\tau = 0.01$
  - Sequence length: $H = 16$
  - Embedding dimension: $d_{model} = 128$
  - Number of attention heads: 4
  - Number of transformer layers: 3
  - Dropout rate: 0.1
  - Weight decay: 1e-5

These hyperparameters were selected based on preliminary experiments, with grid search used to identify optimal values for key parameters like learning rate and network size. This tuning process ensures that each approach performs at its best, enabling fair comparison of their inherent capabilities.

4.2.2 Verification by Simulation

The proposed methods were analyzed by simulation on environment similar to a training one. Both FF-DQN and TE-DDQN have trained three models with 3 different

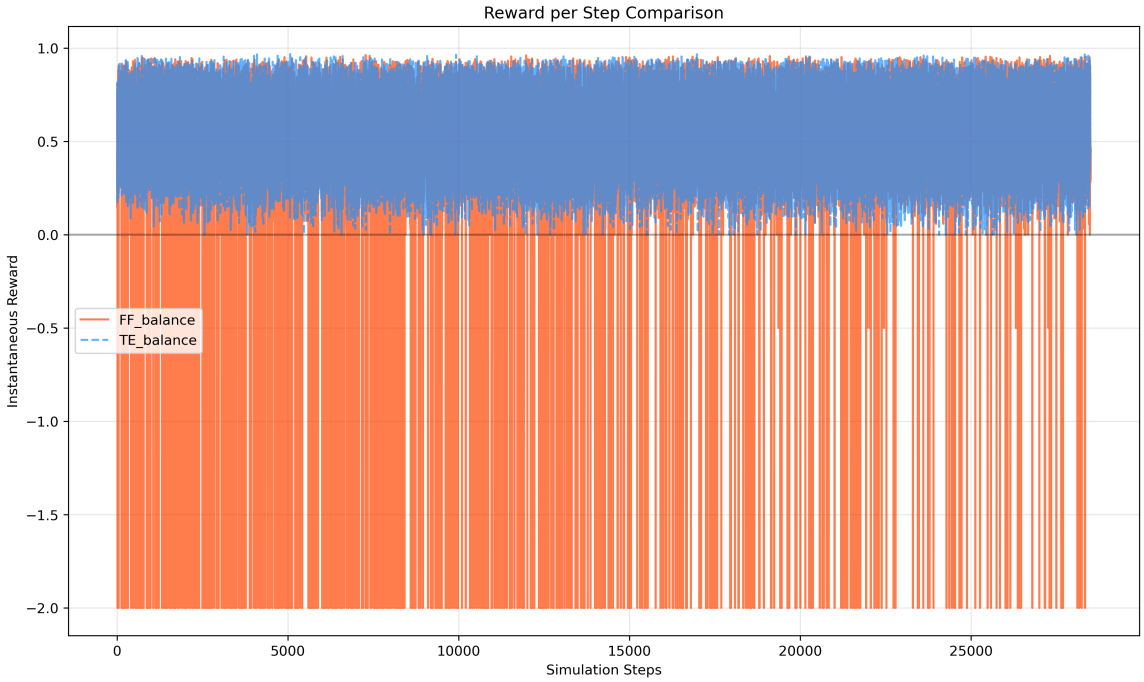

Figure 4-6: Reward fluctuation FF-DQN vs TE-DDQN

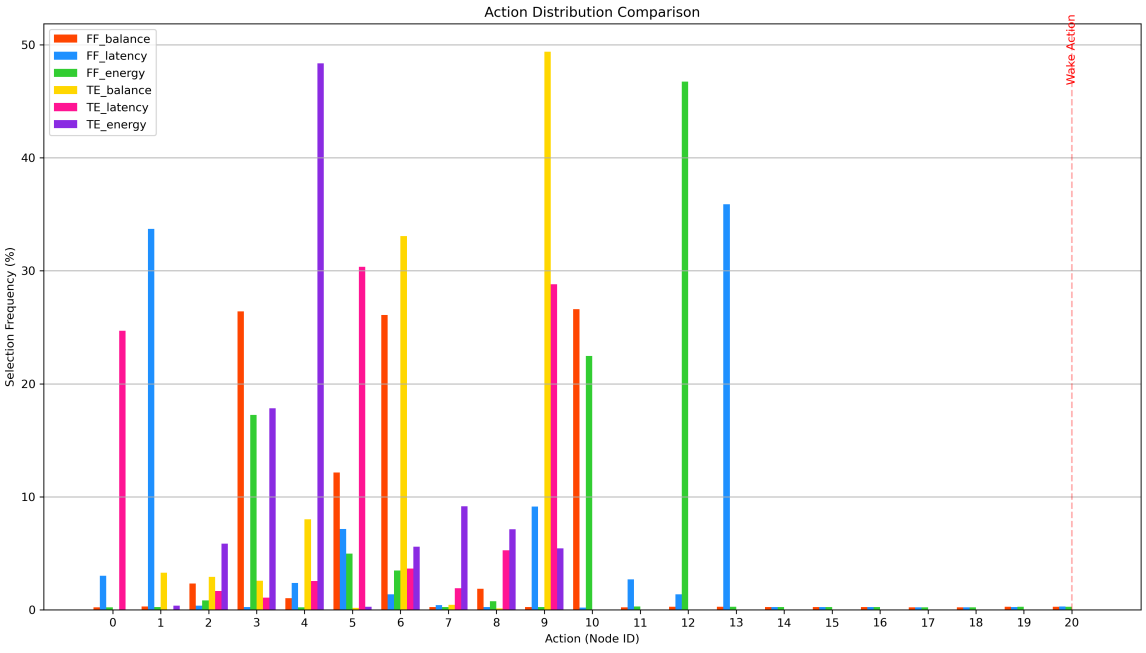

Figure 4-7: Node load distribution comparison

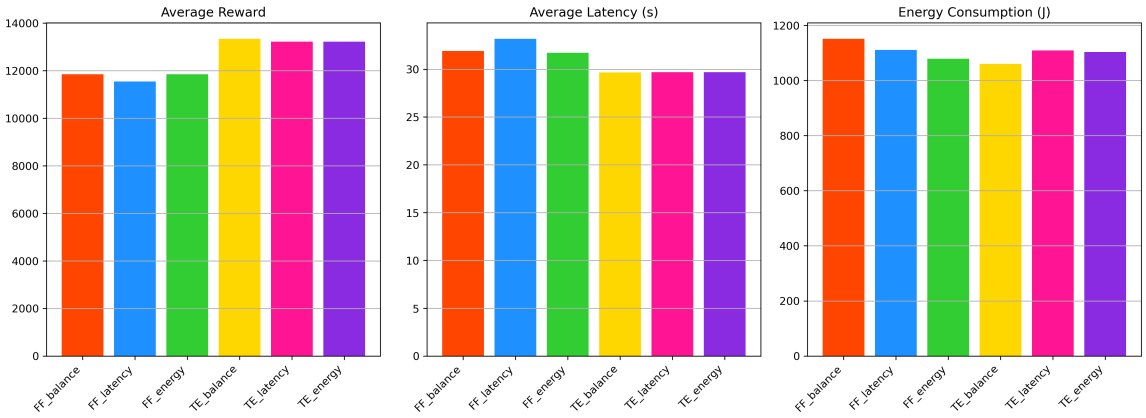

Figure 4-8: Performance of multiple models under Low load

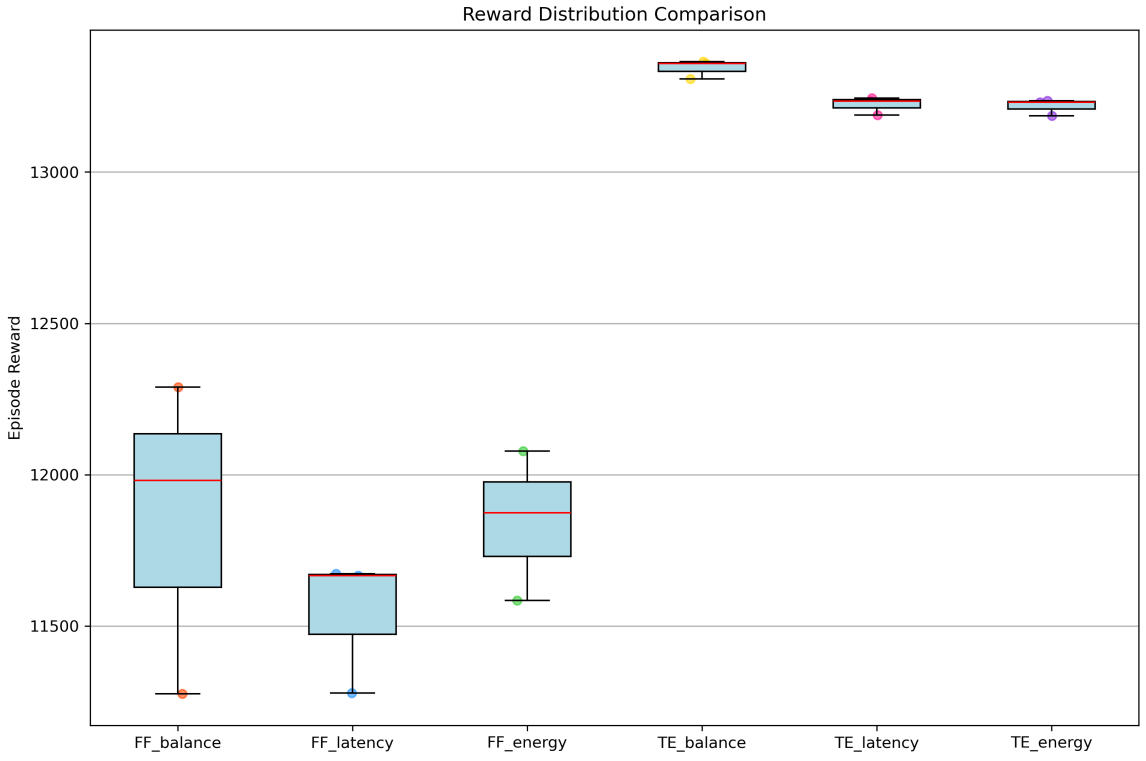

Figure 4-9: Reward distribution of multiple models under Low load

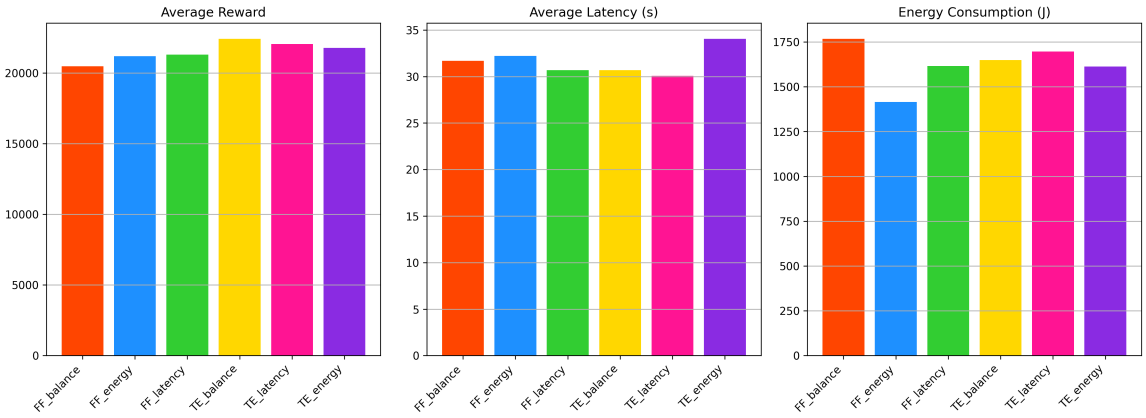

Figure 4-10: Performance of multiple models under Medium load

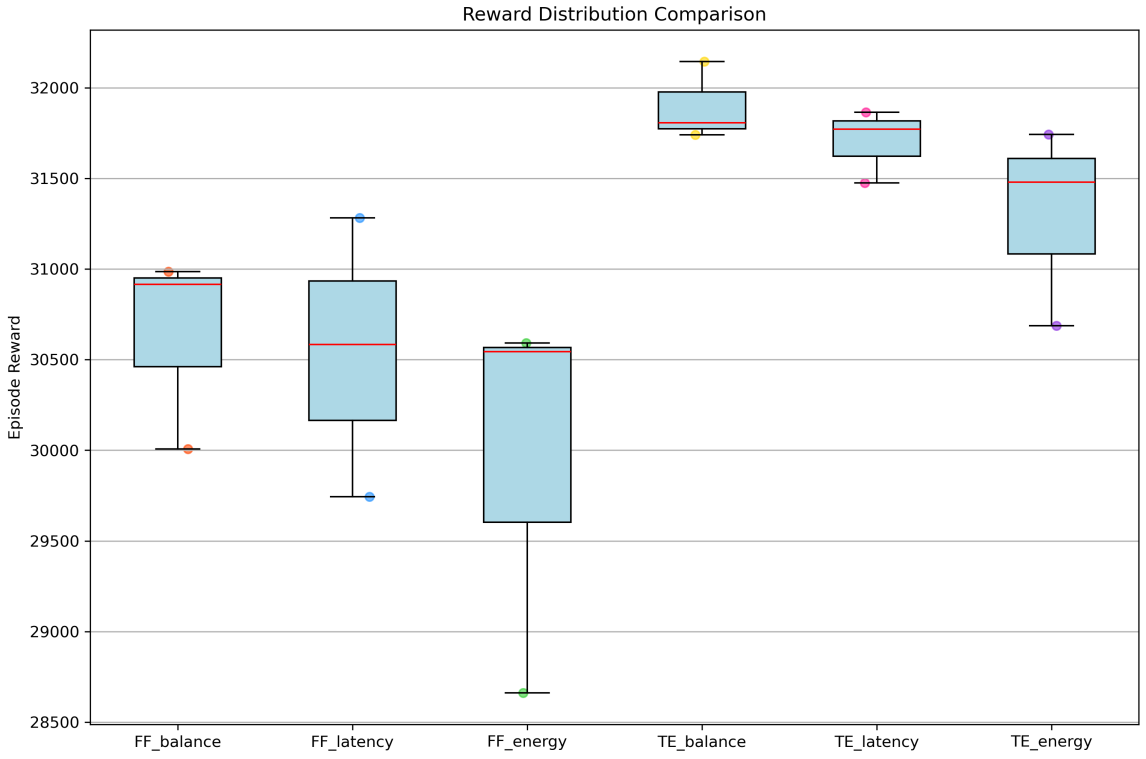

Figure 4-11: Performance of multiple models under Medium load

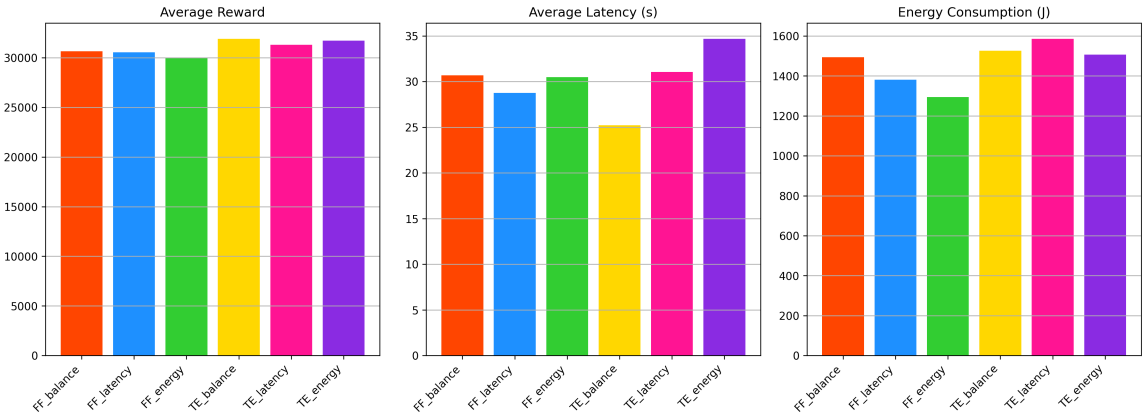

Figure 4-12: Performance of multiple models under High load

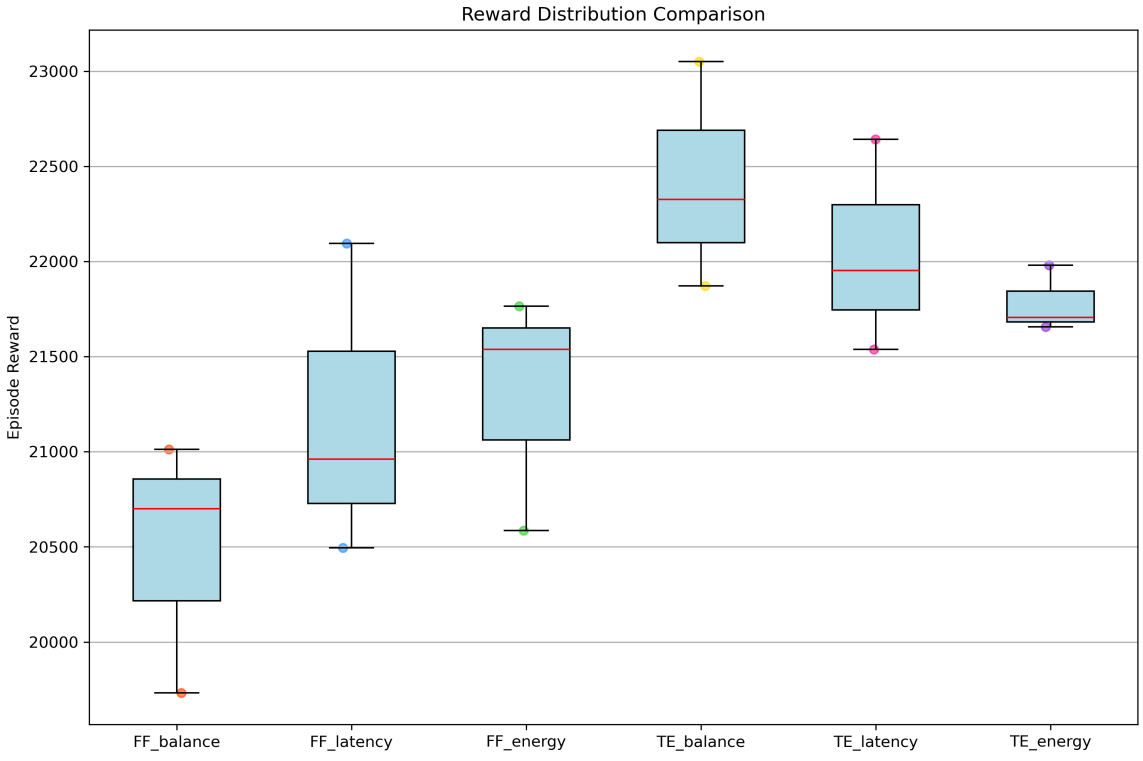

Figure 4-13: Performance of multiple models under High load

reward focus: focus more on latency, focus more on energy consumption and balanced focus. Namely, the parameter Energy weight was set to 0.3, 0.7 and 0.5, respectively.

Also, on these trained 6 models there were three experiments conducted with low, medium and high workload. Low medium had 10 to 15 task offloaded at each step, Medium had workload with 50-60 and high 100 to 120 tasks per step. And Maximum steps per episode was increased to 10,000.

Low Workload

Under light load conditions (see Figure 4-8):

- **TE-DQN** outperformed FF-DQN in reward maximization across all evaluated reward settings.
- TE-DQN achieved lower latency (approximately 29.5 s versus 32 s) and lower energy consumption, while maintaining a higher overall reward.
- The distribution of rewards (Figure 4-9) confirms a tighter and more consistent reward profile for TE-DQN.

Medium Workload

As the task load increased (see Figure 4-10):

- The TE-DQN variant achieved the highest average reward (approximately 32K in the balanced setting) and outperformed all FF-DQN variants.
- The latency-focused TE-DQN variant showed a significant drop in latency (approximately 25 s) with moderate energy savings compared to FF-DQN.
- Figure 4-11 displays a narrower reward variance for TE-DQN, indicating better generalization.

High Workload

Under heavy load conditions (see Figure 4-12):

- Although the reward gap narrowed, TE-DQN still led with an average reward of around 22.8K in the balanced setting compared to approximately 21K for FF-DQN.

- TE-DQN maintained better latency performance (around 30 s) compared to FF-DQN, which often exceeded 32 s.

- The reward distribution (Figure 4-13) further confirms the superior robustness of TE-DQN under stress.

The TE models consistently outperform FF models in terms of overall reward, with TE models achieving approximately 13% higher rewards. Among configuration types, the balanced (0.5) weight demonstrates superior performance for TE models, while the energy-focused (0.7) configuration performs best for FF models.

TE models achieve consistently lower latency values (approximately 29.7s) compared to FF models (approximately 31.6s). The balanced TE configuration yields the lowest latency, suggesting that extreme weighting toward either energy or latency may not be optimal for latency reduction.

FF models with energy-focused configuration (0.7) achieve the lowest energy consumption, while TE models with balanced configuration (0.5) demonstrate the best energy efficiency among TE variants. This indicates different optimal weight configurations exist between model architectures.

The action distribution reveals distinct node selection strategies:

- FF models exhibit more dispersed action patterns, with latency-focused models favoring nodes 1, 2, and 14

- TE models show more concentrated selection patterns, with clear preferences for nodes 4, 6, and 10

- The energy-focused TE model uniquely favors node 4 (48% of decisions)

- Wake-up actions (node 20) are minimal across all models

The Transformer-Enhanced architecture demonstrates superior overall performance with more stable reward distributions and lower latency. However, model architecture interacts significantly with energy-latency weighting parameters. The FF architecture may offer better energy efficiency when specifically optimized for energy, while TE models provide better general performance and latency characteristics regardless of weighting.

4.2.3 Physical Testbed Evaluation

To validate our simulation results in a real-world environment, we conducted experiments on a physical testbed utilizing the RAPID framework with a Jetson Nano cluster. The training environment was modified by reducing the number of base stations to one to align with our physical testbed configuration. Both FF-DQN and TE-DDQN models were trained with a maximum task parameter of 5-6 to match the arrival rate achievable on the physical system.

Both models converged as expected, as illustrated in Figures 4-15 and 4-16, with FF-DQN exhibiting more volatile training loss and slower convergence compared to TE-DDQN. Under evaluation conditions similar to the simulation experiments, TE-DDQN demonstrated clear superiority over FF-DQN with significantly higher rewards (2100 compared to FF-DQN's 1350), lower power consumption, but slightly higher latency under the low workload condition of 10-15 tasks per step (Figure 4-17).

The workload was modeled using a Poisson distribution (Figure 4-14) featuring two peak periods. This workload had the following characteristics: mean inter-arrival time of 0.471s, variance of inter-arrival times of 0.942, and a total of 1,269 tasks over a 10-minute period.

The physical experiment results revealed comparable performance patterns between the two approaches. As shown in Figures 4-18 and 4-19, FF-DQN completed processing later, resulting in higher cumulative energy consumption despite having lower average turnaround time due to its faster inference capabilities. This extended processing time may be attributed to FF-DQN occasionally selecting powered-off nodes (similar to patterns observed in Figure 4-7).

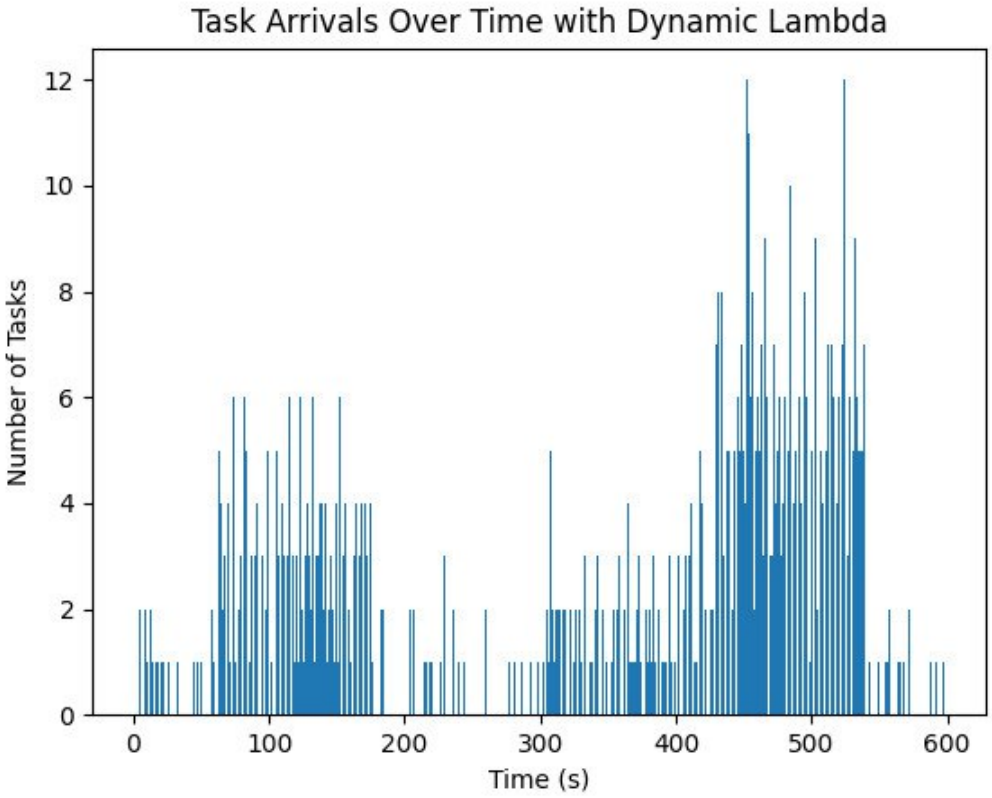

Figure 4-14: Arrival rate of tasks as Poisson process

The overall performance comparison depicted in Figure 4-20 shows comparable average power consumption between the two approaches, consistent with the results in Figure 4-18. However, the difference in turnaround time became more evident in the physical testbed. Interestingly, while TE-DDQN utilized less memory on average, this did not significantly impact overall performance metrics.

These physical testbed results validate our simulation findings and demonstrate that the TE-DDQN approach maintains its advantages in real-world deployment scenarios, particularly in terms of reward maximization and energy efficiency.

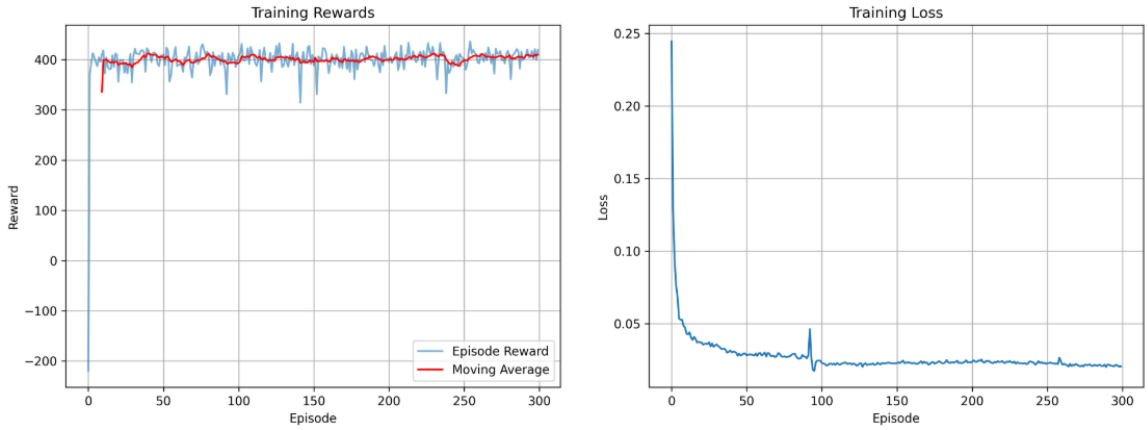

Figure 4-15: TE-DDQN training results

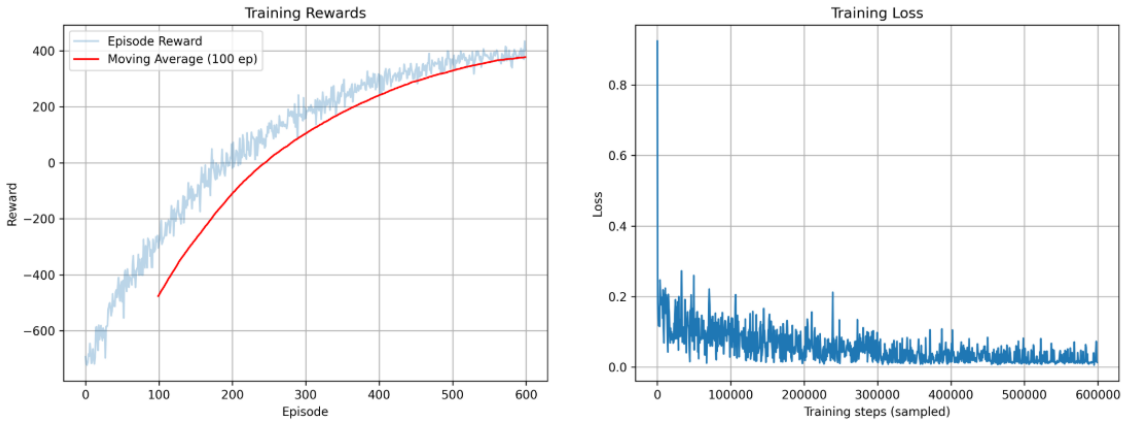

Figure 4-16: FF-DQN training results

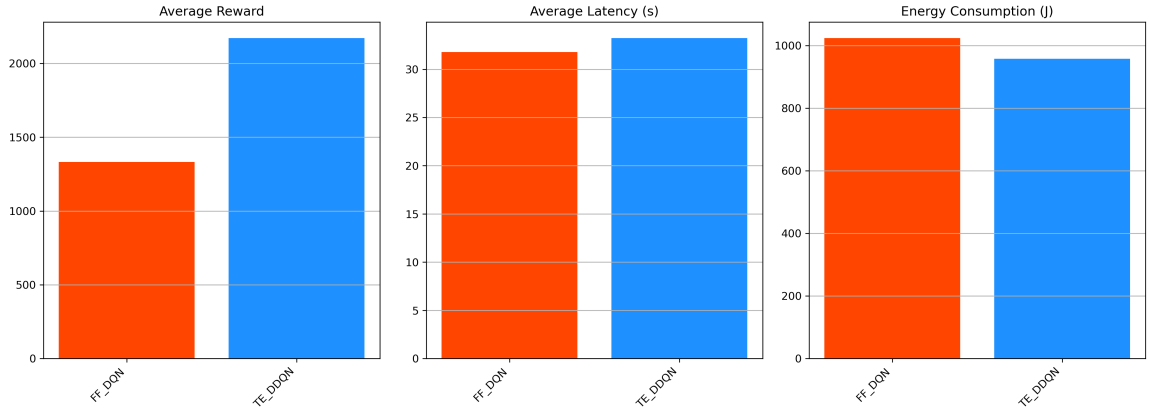

Figure 4-17: Simulation results for physical testbed models

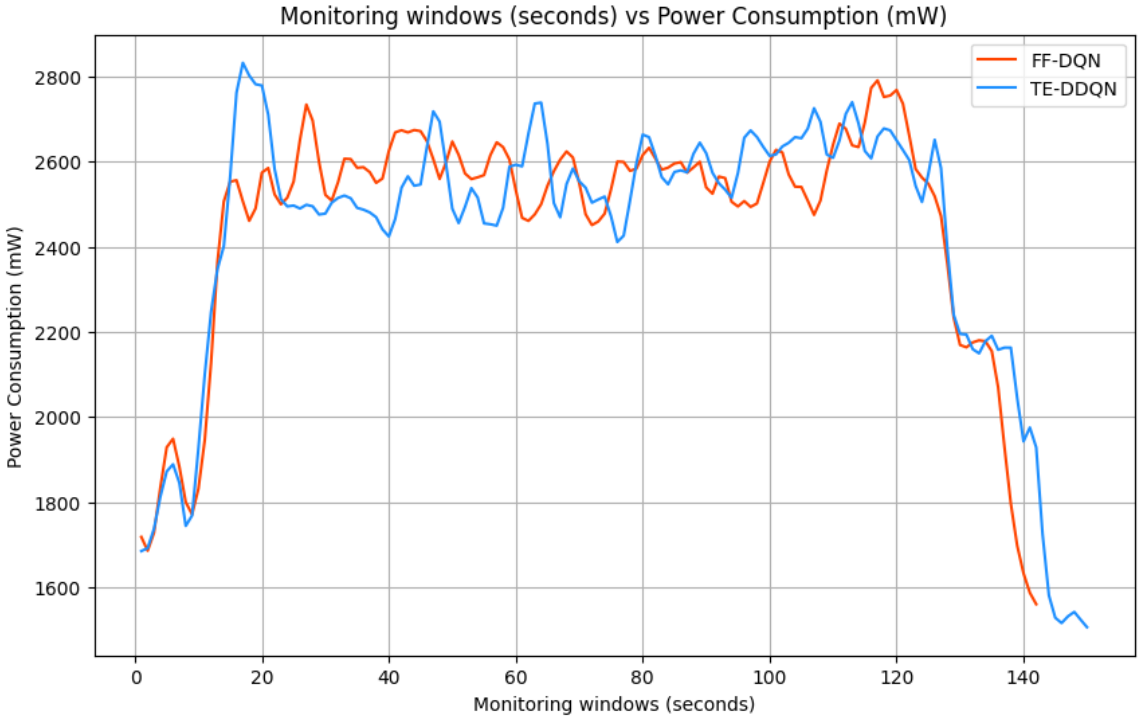

Figure 4-18: Energy consumption results on physical testbed

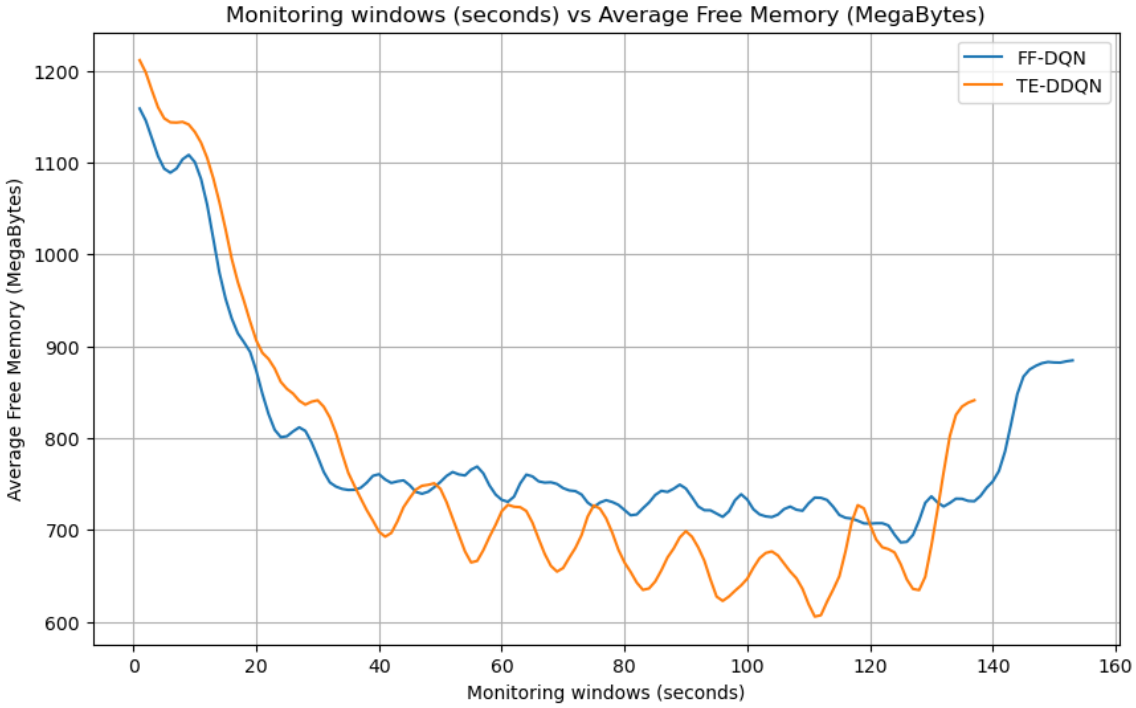

Figure 4-19: Average free memory results on physical testbed

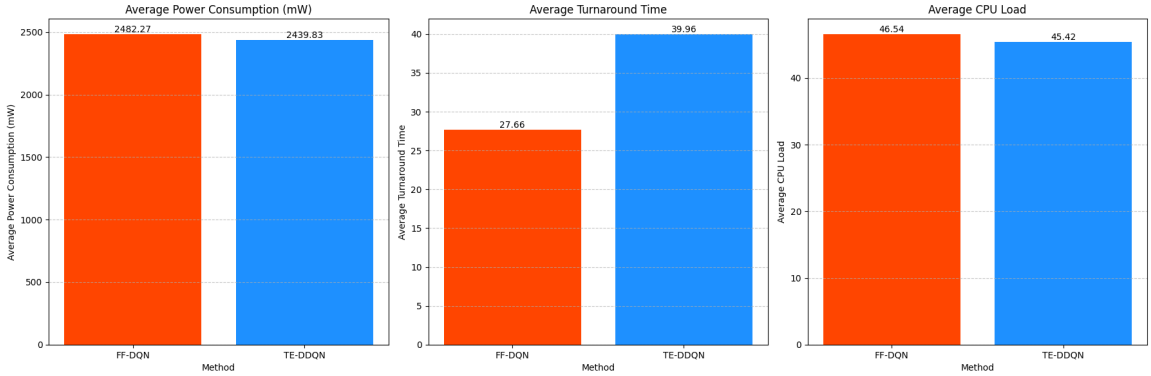

Figure 4-20: Summary of physical testbed results

Chapter 5

Conclusion

5.1 Limitations and Future Work

While our experimental analysis demonstrates the overall advantage of the Transformer-based approach for Reinforcement Learning solution to Vehicular Edge Computing task offloading, several limitations and opportunities for future work remain:

Even though our Results demonstrate the clear superiority of Transformer-based solutions, several limitations can be addressed in future work including:

Sequence Length Sensitivity: The performance of the TE-DDQN is sensitive to the chosen sequence length. Our implementation uses a fixed length of 16, however, there is a possibility that adaptive sequence length might further improve performance both in convergence speed and performance results.

Heterogeneous Computing Resources: Current environment assumes a homogeneous edge cluster consisting only of Nvidia Jetson-Nano devices. Extending the approach to heterogeneous resources with varying computational capabilities such as adding other energy-efficient and highly computationally capable nodes such as Jetson Xavier or Orin might get closer to a real heterogeneous environment.

Environmental Variability: Despite having heterogeneous vehicular environment with real-world elements such as map of Astana city, variety of offloading tasks and empirical energy model, it does not fully encompass all possible scenarios and variables that might real-world environment bring.

Future Work Opportunities:

- Implementing ablation studies to quantify the contribution of each component
- Extending the model to handle more complex mobility patterns and network topologies
- Incorporating more sophisticated energy models that account for dynamic voltage and frequency scaling
- Exploring multi-agent framework to reduce time taken for decision inference

5.2 Conclusion

In this paper, this work presented an energy and latency-efficient task offloading approach for VEC environment, with a particular incline towards real-world environment elements. Two reinforcement learning solutions were compared: a standard FF-DQN approach and a state-of-the-art TE-DDQN solution.

The proposed methodology incorporated real-world energy measurements from Jetson Nano devices, and realistic vehicular mobility patterns using SUMO simulation with a map of Astana city. The results demonstrated that the TE-DDQN approach consistently outperformed the standard DQN, achieving higher task completion rates, lower latency, and reduced energy consumption. As well as especially notable faster convergence, and more stable performance.

The key insights from our work include:

- The value of integrating real-world energy measurements into the RL training environment
- The effectiveness of dynamic node management with Wake-on-LAN capability for energy efficiency
- The complexity of the inference of the decision model is crucial in real-world testbed

- The importance of temporal pattern recognition in the VEC environment
- Both approaches ultimately achieve similar reward scales during training, but the Transformer model demonstrates better stability and closer to optimal energy consumption and latency.

This work contributes to advancement of development and integration of RL models to such heterogeneous environments such as VEC. The obtained performance measurements of Transformer based solution opens promising solution for edge computing environments.

Bibliography

[1] A. Karzhaubayev M. Murzabulatov A. Madiyev, D. Bulegenov and D. M. Bu. Energy-efficient offloading framework for mobile edge/cloud computing based on convex optimization and deep q-network. *Research Square (Preprint)*, 2025. Version 1, posted April 2, 2025.

[2] A. Karzhaubayev M. Murzabulatov A. Madiyev, D. Bulegenov and D. M. Bui. Energy-efficient scheduling and optimization in edge computing based on deep q-networks. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 00827–00833, Las Vegas, NV, USA, 2025.

[3] A. B. De Souza et al. Computation offloading for vehicular environments: A survey. *IEEE Access*, 8:198214–198243, 2020.

[4] B. Lin et al. Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics*, 2020.

[5] B. Lin et al. Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics*, 2020.

[6] C. Zeng et al. Joint optimization of multi-dimensional resource allocation and task offloading for qoe enhancement in cloud-edge-end collaboration. *Future Generation Computer Systems*, 2024.

[7] H. Seo et al. Differential pricing-based task offloading for delay-sensitive iot applications in mobile edge computing system. *IEEE Internet of Things Journal*, 2022.

[8] J. Wang et al. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[9] J. Wang et al. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[10] K. Yang et al. A novel distributed task scheduling framework for supporting vehicular edge intelligence. In *IEEE International Conference on Distributed Computing Systems*, 2022.

[11] L. Chen et al. Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation. *IEEE Transactions on Cloud Computing*, 2022.

[12] L. Zhao et al. Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing. *IEEE Transactions on Mobile Computing*, 23(5):4259–4271, 2024.

[13] M. Xue et al. Eosdnn: An efficient offloading scheme for dnn inference acceleration in local-edge-cloud collaborative environments. *IEEE Transactions on Green Communications and Networking*, 2022.

[14] N. Abbas et al. Joint computing, communication and cost-aware task offloading in d2d-enabled het-mec. *Computer Networks*, 2022.

[15] S. Li et al. Energy-efficient task offloading using dynamic voltage scaling in mobile edge computing. *IEEE Transactions on Network Science and Engineering*, 2021.

[16] S. Liu et al. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE Journal on Selected Areas in Communications*, 2023.

[17] T. Do-Duy et al. Digital twin-aided intelligent offloading with edge selection in mobile edge computing. *IEEE Wireless Communications Letters*, 2022.

[18] T. Ji et al. Energy-efficient computation offloading in mobile edge computing systems with uncertainties. *IEEE Transactions on Wireless Communications*, 2022.

[19] V. Cozzolino et al. Nimbus: Towards latency-energy efficient task offloading for ar services. *IEEE Transactions on Cloud Computing*, 2023.

[20] X. Li et al. Energy-efficient computation offloading in vehicular edge cloud computing. *IEEE Access*, 2020.

[21] Y. Wang et al. Computation offloading optimization for uav-assisted mobile edge computing: a deep deterministic policy gradient approach. *Wireless Networks*, 2021.

[22] Askar Madiyev. Thesis code repository. https://github.com/Askar147/Thesis, 2025. Accessed: 2025-04-11.

[23] Wolfram Schneider. Bbbike extracts: Openstreetmap data extract service. https://extract.bbbike.org/, 2025. Accessed: 2025-04-11.

[24] X. Sun, Y. Hu, and M. Chen. Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments. *IEEE Internet of Things Journal*, 9(5):10243–10254, 2022.

[25] M. Tang and V. W. S. Wong. Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, 2020.

[26] S. Wang and T. He. Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 18(1):85–98, 2021.

[27] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian. Computation offloading in multi-access edge computing: A multi-task learning approach. *IEEE Transactions on Mobile Computing*, 20(9):2745–2762, 2021.