Task Management Optimization in Vehicular Edge Computing

by

Mahsa Paknejad

A thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the

Master of Applied Science
Electrical and Computer Engineering
Concentration in Applied Artificial Intelligence

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

External Member(s):

Jie Gao
Assistant Professor,
School of Information Technology,
Carleton University

Internal Member(s):

Abdulmotaleb El Saddik
Professor,
School of Electrical Engineering & Computer Science,
University of Ottawa

Supervisor(s):

Hussein T. Mouftah
Distinguished University Professor,
School of Electrical Engineering & Computer Science,
University of Ottawa

Burak Kantarci
University Research Chair, Professor,
School of Electrical Engineering & Computer Science,
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Vehicular Edge Computing (VEC) is a specialized extension of Mobile Edge Computing (MEC) designed to support real-time processing in intelligent transportation systems. It enables vehicles to offload computationally intensive tasks to nearby Roadside Unit (RSU)s equipped with MEC servers, reducing latency for time-critical applications such as autonomous driving, intelligent traffic control, and Vehicle-to-Everything (V2X) communication. However, the mobility of vehicles and the short coverage duration within an RSU's range present significant challenges for task offloading and scheduling. The growing volume of tasks from multiple vehicles can cause congestion, delays, and task drops, undermining overall system performance. To establish a performance benchmark, we first employ deterministic algorithms, including First-Come, First-Served (FCFS) and Shortest Deadline First (SDF), which are simple and fast methods for real-time task offloading. To explore more efficient scheduling, we then apply a metaheuristic method called Particle Swarm Optimization (PSO). This optimization approach is first evaluated in a static scheduling environment, in which the algorithm is executed only once on all tasks, without considering their real-time arrivals, and schedules them all at once. FCFS executes tasks in arrival order without prioritization, while SDF improves performance by prioritizing tasks with shorter deadlines. PSO achieves the best performance in this static environment because execution time and additional waiting times caused by the static nature are ignored. This method is called Offline Static PSO (Off-Sta-PSO), and it provides the theoretical upper bound. By considering the real-time execution and waiting times introduced in this scenario, we also establish the lower bound case, called Online Static PSO (On-Sta-PSO), which yields the worst performance. Recognizing the limitations of task offloading, we also propose a task partitioning approach. Some portions of the tasks are offloaded to RSUs, while the remaining parts are processed locally by onboard vehicle processors, leading to reduced latency and fewer dropped tasks. Moving to real-time task offloading, while dynamic PSO enhances task scheduling, its high computational cost and long convergence times limit its real-time viability. To address this, we introduce Online Dynamic Cost-Driven Algorithm (On-Dyn-CDA), a novel real-time scheduling algorithm. Unlike PSO, it operates in milliseconds, adapts to vehicle mobility, congestion, and RSU load, and requires no pre-training. On-Dyn-CDA surpasses Dynamic PSO by 3.42% in task loss, reduces latency by 29.22%, and executes in just 0.05 seconds under the most complex scenario, compared to 1330.05 seconds required by Dynamic PSO. Finally, we compare online PSO with Deep Reinforcement Learning (DRL) methods like Deep Q-Network (DQN) and proximal policy optimization (PPO). Although Reinforcement Learning (RL) is grounded in the Markov Decision Process (MDP), which assumes a stationary environment, VEC systems are inherently dynamic due to vehicle mobility, changing task arrivals, and vari-

able RSU associations. To address this mismatch, our framework introduces a decision window mechanism that segments incoming tasks into locally stable intervals, allowing the RL agent to operate under near-stationary conditions. Additionally, we design adaptive reward functions that guide the agent to minimize both task drops and end-to-end (E2E) latency, based on real-time task characteristics and server availability. The state space includes dynamic context such as MEC server availability and task deadlines, enabling informed and responsive decision-making. Our DQN and PPO models are trained on diverse mobility traces and evaluated in unseen environments, demonstrating strong generalization without retraining. Despite the non-stationarity of VEC, this design enables robust online scheduling, with DQN outperforming dynamic PSO in both latency and task reliability. DQN substantially reduces execution time, completing in only 10.62 seconds, lowers dropped tasks by 2.5%, and decreases E2E latency by 18.6%. Compared to PPO, DQN achieves a 57.1% reduction in execution time, along with a 5.7% decrease in E2E latency and a 1.7% reduction in dropped tasks. The results demonstrate the effectiveness of this research in addressing the core challenge of real-time task scheduling in VEC systems. [1]

[1] Text shown in blue throughout this thesis indicates hyperlinks to external or internal sources.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Professor Hussein T. Mouftah and Professor Burak Kantarci, for their continuous support, guidance, and belief in my potential throughout every step of my academic journey. I am truly fortunate to have had the opportunity to learn from both of them. Professor Mouftah's kindness and care, like that of a father, and his consistent encouragement have played a vital role in my development, giving me both confidence and direction. At the same time, Professor Kantarci's trust in me and his support in providing access to his laboratory, located in the largest technology park in Canada, offered me a unique opportunity to engage with industry-driven projects and gain invaluable hands-on experience. He has further contributed to my growth by providing me with various opportunities that have shaped both my academic and personal development.

I would also like to thank Dr. Murat Simsek for his invaluable guidance and support throughout this project. His thoughtful mentorship, patience, and genuine willingness to help were instrumental in my progress. He consistently made time to discuss ideas, clarify doubts, and provide constructive feedback. Our regular brainstorming sessions and in-depth discussions not only kept me focused but also encouraged me to approach problems from new perspectives.

A special thank you goes to Parisa Fard Moshiri, who has been more like a kind-hearted sister than a friend. Her generous help, shared experience, and constant support have meant a great deal to me during this journey.

Last but not least, I am deeply thankful to my parents, who made great sacrifices to allow me to pursue my studies far from home. Their love and support are the foundation of all my achievements. I also extend my heartfelt appreciation to my uncle and his family, who treated me like their own daughter and supported me in countless ways while I was away from home, which allowed me to concentrate more fully on my studies.

Table of Contents

List of Tables

List of Figures

Glossary

downlink Receive the processed results back to the vehicle. 3, 35, 37–41, 59, 80

DTMS A first-come-first-serve based scheduling algorithm, which assigns the same priority to all users without considering their heterogeneous delay requirements and channel conditions [68]. 19

IHRA A branch-and-bound based heuristic method, which intends to minimize the offloading delay under the cooperation of edge computing and cloud computing [68]. 19

MEC+Local Partitioning. xi, 34–38, 42, 44–46

MECO A method to optimize resource allocation based on OFDMA by considering the priorities, energy consumption and channel conditions of vehicular users [68]. 19

MGA An improved scheduling scheme based on a genetic algorithm [114]. 28

MMTO-O A scenario when tasks are offloaded to only one hop vehicles in MMTO [48]. 18

particle Candidate solution in Particle Swarm Optimization. 4

RPA-O A scenario when tasks are offloaded randomly to only one hop vehicles in RPA [48]. 18

SA-DQN An offloading scheme based on DQN [114]. 28

TOD A method which selects the network based on an approach belonging to the UCB family, namely, discounted-UCB [8]. 32

uplink Transmitting task data from the vehicle to the edge server. 3, 35, 37, 39–41, 59, 80

Abbreviations

A2C Actor-Critic 31

A3C Asynchronous Advantage Actor-Critic 23, 31

AC Actor-Critic Algorithm 29

ACO Ant Colony Optimization 50

ACTO-n Actor–Critic Task Offloading 24, 31, 75, 76

AdaUCB Adaptive Upper-Confidence-Bound 27

ADRO Adaptive Data Rate-based Offloading algorithm 13, 19

AEC All Edge Computing 15

ALC All Local Computing 15

ALTO Adaptive Learning-Based Task Offloading 27

AODAI Advantage-Oriented Dueling Actor-Insulator Network 24, 31, 75, 76

APA Local Processing Algorithm 18

APSO Adaptive Particle Swarm Optimization 19, 27

BBO Bound-and-Bound based Optimal 32

BME Bayesian Maximum Entropy 9

BS Base Station 51

BSs Base Stations 13, 23, 25

BTV Bee colony-based Task offloading in Vehicular edge computing 10, 16

CAVs Connected and Autonomous Vehicles 34, 48

CC Cloud Computing 13

CNN Convolutional Neural Network 21

CS Cuckoo Search 17

CTO Centralized Task Offloading 27

DAG Directed Acyclic Graph 8

DAGGEN Directed Acyclic Graph Generator 29, 50, 75

DAGs directed acyclic graphs 21, 23, 49, 52

DDPG Deep Deterministic Policy Gradient 21, 23, 24, 26, 31, 32, 36

DDPGTO Deep Deterministic Policy Gradient-based Task Offloading Algorithm 33

DDQN Double Deep Q-network 20, 22, 23, 25, 27, 29, 30, 32

DE Differential Evolution 31

DEFO Distributed Earliest-Finish Time Offloading 29

DFO Dynamic Framing Offloading 32

DGO Delay-Greedy Optimization 31

DIOW Distributed Intelligent Task Offloading and Workload Balance 23, 31

DND Department of National Defence 46, 72, 88

DNN Deep Neural Network 5, 25, 32

DOS Dynamic Task Offloading Strategy 12, 17

DQN Deep Q-Network iv, v, x, 5, 6, 19, 20, 22–25, 29–33, 73–76, 84–88, 90, 91

DRL Deep Reinforcement Learning iv, 20–23, 25, 32, 36, 50, 51, 73

DTOSC Dependency-Aware Task Offloading and Service Caching 11, 17

E2E end-to-end v, xi, 6, 8, 13, 34, 40, 42, 43, 45, 46, 56, 60, 67, 70, 73, 74, 76, 80, 85, 86, 88, 90

EAS Equal Allocation Strategy 18

EMT Multi-Vehicle Task Offloading 13, 18

ETSI uropean Telecommunications Standards Institute 1

FCFS First-Come, First-Served iv, xi, 3, 19, 20, 27, 33, 34, 37, 46, 55, 67–72, 87, 90

FDTO fully distributed task offloading 20, 27

FIFO First In, First Out 16

FL-TD3 Federated Learning-Twin Delayed Deep Deterministic Policy Gradient 24, 32

FREGA Fuzzy Relative Entropy Genetic Algorithm 15

FRL Federated Reinforcement Learning 19, 27

GAN Generative Adversarial Network 22

GCNs Graph Convolutional Networks 20

GEPAGA Grey Entropy Parallel Analysis 15

GRLVTO Graph-powered Reinforcement Learning for Vehicular Task Offloading 28

GTT greedy task by task 16

GWO Grey Wolf Optimizer 17

HGOS Heuristic Greedy Offloading Scheme 28

HJTR Heuristic Joint Task offloading and Resource allocation algorithm 75

ICRI Integer Constraint Relaxation Iterative 75

IDEaS Innovation for Defence Excellence and Security 46, 72, 88

IoT Internet of Things 1, 48

IoV Internet of Vehicles 11, 49, 75, 76

JTOS Joint Task Offloading and Scheduling Algorithm 18

LBA Local Based Algorithm 30

LBPM Lyapunov-Based Profit Maximization 11, 16

LE Local Execution 29

LEGAP "Less than or Equal to" Generalized Assignment Problem 26

LODCO Lyapunov Optimization-Based Dynamic Computation Offloading 18

LOS Local Offloading Strategy 17

LRU Least recently used 31

LSTM long short-term memory 50, 51

MAB Multi-armed Bandit 27

MACTER Mobility-Aware Computational Efficiency-Based Task Offloading and Resource Allocation 11, 16

MADDPG Multi-Agent Deep Deterministic Policy Gradien 23, 50, 51

MAML model-agnostic meta-learning 76

MAPPO-TTSA Multi-Agent Proximal Policy Optimization-Based Task and Target Selection Algorithm 29

MCLA Mobility, Contact, and Computational Load-Aware 50

MDO Multidecision Based Offloading 16

MDP Markov Decision Process iv

MDPs Markov Decision Processes 76

ME MEC Server Execution 29

MEC Mobile Edge Computing iv, v, xi, 1, 2, 6, 8–13, 15, 16, 19, 22, 24, 25, 30, 34–52, 55–58, 60–66, 73–76, 78–83

MEC-V2X MEC for Vehicle-to-Everything 16, 75

MEGAN Mobile Edge Computing Assisted Task Offloading using Generative Adversarial Network 22, 30

MESON Mobility-aware dependent task offloading 20, 21, 28

Meta-RL Meta Reinforcement Learning 21, 23, 75, 76

MINLP Mixed-Integer Nonlinear Programming 11, 23

ML Machine Learning vii, 4, 5, 19, 48, 52, 72, 74

MLDBA Multi-Layer D3QN-Based Algorithm 30

MLDPBA Multi-Layer D3QN and PDQN-Based Algorithm 30

MMTO Mobility-Aware Multi-hop Task Offloading 18

MOEA/D Multi-Objective Evolutionary Algorithm based on Decomposition 15

MOHGA Multi-Objective Hybrid Genetic Algorithm 9, 15

MPTO Meta-Policy based Task Offloading 30

MTO multi-task offloading 52, 76

NGTO Noncooperative Game based Task Offloading 27

NM Nelder-Mead 31

NO Nearby offloading 18

NoBP No BS Peer Offloading 31

NOMA Non-Orthogonal Multiple Access 10, 13, 15, 36, 51

NSERC Natural Sciences and Engineering Research Council of Canada 46, 72, 88

NSGA Non-ominated Sorting Genetic Algorithm 15

NSO Node select optimization 18

ODCO Optimized Distributed Computation Offloading 21, 29

OFDMA Orthogonal Frequency-Division Multiple Access 36

Off-Sta-PSO Offline Static PSO iv, 19, 33, 49, 50, 55, 67, 68, 70, 76, 84

OJTR Optimal Joint Task offloading and Resource allocation algorithm 75

On-Dyn-CDA Online Dynamic Cost-Driven Algorithm iv, xi, 19, 33, 47–50, 52, 56, 63–65, 67, 68, 70–72, 90, 91

On-Dyn-PSO Online Dynamic PSO 19, 33, 50, 55, 56, 68, 70, 72, 76, 85–87, 90, 91

On-Sta-PSO Online Static PSO iv, 19, 33, 49, 50, 55, 67, 68, 70

ORF-RE Ontario Research Fund-Research Excellence 88

PBLA Probability-Based Location Aware 14, 50

PBTSA Priority-Based Task Scheduling Algorithm 8, 14, 49, 50

PDAGTO Priority and Dependency-Based DAG Tasks Offloading 17

POETS Partial computation Offloading and adaptive Task Scheduling algorithm 13, 19

POETS w.o. NOMA POETS without NOMA access 19

PPO proximal policy optimization iv, v, x, 5–7, 19, 23, 29, 30, 33, 50, 51, 73–76, 84–88, 90, 91

PSO Particle Swarm Optimization iv, v, x, xi, 4, 6, 9, 10, 19, 34, 36–38, 42–49, 52, 55, 56, 63, 66–68, 70–76, 83, 86, 87, 90, 91

PTORA particle swarm optimization based task offloading and resource allocation 9, 15

PTR Particle swarm optimization-based Task offloading and Resource allocation 10, 15

QoS Quality of Service 43

R2C Revenue-to-Cost 32

RALPTO RSU-assisted Learning-Based Task Offloading Algorithm 28

RAS Random Allocation Strategy 18

RBA Random Based Algorithm 30

RBFS Reverse Breadth-First Search 8, 49

Rf-n Reinforce-based approach 31

RL Reinforcement Learning iv, v, vii, xii, 5, 6, 19, 20, 22–26, 50–52, 72, 74–76, 81–84, 86–91

RMOS Random MEC Server for Offloading Strategy 17

RO Random Offloading 29

RPA Random Policy Algorithm 18

RSU Roadside Unit iv, v, 2, 8, 10, 13, 19–21, 25, 35, 37–41, 43, 47, 48, 51, 55, 57, 58, 60, 65, 66, 73–75, 78, 79, 83, 90

RWGA Random Weighting Genetic Algorithm 15

SAC Soft Actor-Critic 24, 32

SARSA State Action Reward State Action 31, 75

SDF Shortest Deadline First iv, xi, 3, 19, 33, 34, 37, 46, 55, 67–72, 87, 90

SDN Software-Defined Networking 9, 11, 12

Seg Task Segmentation 32

SIC Successive Interference Cancellation 10

SL Shortest Latency 31

SLDBA Single-Layer Deep-Based Algorithm 30

SMRL-MTO Seq2Seq- based Meta Reinforcement Learning 29, 50, 51, 75

SPEA strength Pareto-based evolutionary algorithm 15

ST Shortest Tolerance Time 31

SUMO Simulation of Urban Mobility 16, 18, 19, 28, 33, 37, 43, 50, 55, 56, 65, 75, 76, 83

SVs Service Vehicles 13

SW-UCB Sliding-Window UCB 32

TD3 Twin Delayed Deep Deterministic Policy Gradient 24, 32

TNP Task Native Processing 28

TOACO Task Offloading based on Ant Colony Optimization 17

TOSM Task Outsourcing Then Relocation 28

TOSO Task Outsourcing Only 28

TOWSC Task Offloading without Service Cache 17

TOWTU Task Offloading without Task Urgency 17

UCB Upper-Confidence-Bound 27, 28, 32

V2I Vehicle-to- Infrastructure 10, 26, 36, 50

V2V Vehicle-to-Vehicle 10, 13, 16, 19, 26, 33, 36, 48, 50

V2X Vehicle-to-Everything iv, 48

VC Vehicular Cloud 24

VEC Vehicular Edge Computing iv, v, vii, xi, 1–5, 8, 10–13, 18–21, 23, 24, 26, 34, 35, 46, 48, 53, 73, 89–91

VECC Vehicular Edge–Cloud Computing 23

VEoTC Vehicle Edge of Things 13, 18

VGWO variant grey wolf optimizer 12, 17

VUCB volatile UCB 28

VUEs Vehicular User Equipments 51

WOA Whale Optimization Algorithm 17

WSPT Weighted Shortest Processing Time 12

Chapter 1

Introduction

MEC allows mobile devices to offload heavy computational tasks to nearby edge servers, improving response time and conserving energy, which is essential for real-time applications like autonomous driving. However, MEC faces challenges such as high loads of tasks on the servers, leading to increased waiting times and dropped tasks. This thesis investigates solutions to these challenges within the context of vehicular networks.

MEC is a network architecture concept introduced by the uropean Telecommunications Standards Institute (ETSI) around 2014. It was developed to address the growing demand for low-latency, high-bandwidth services and to support the increasing number of connected devices in mobile networks. The core idea of MEC is to bring cloud-computing capabilities closer to end-users by deploying computing and storage resources at the edge of the mobile network, such as base stations or access points [28]. By processing data near its source, MEC reduces the need to send information to distant cloud data centers, leading to faster response times, reduced backhaul traffic, and more efficient network usage. MEC was introduced as a response to the limitations of traditional cloud computing in handling real-time, location-aware, and bandwidth-sensitive applications, and it has since become a foundational technology for emerging fields like the Internet of Things (IoT), augmented reality, and connected vehicles [21].

To reflect its applicability beyond mobile networks, the term Multi-access Edge Computing is adopted to encompass other access technologies such as Wi-Fi, fixed broadband, and satellite. This broader term highlights MEC's flexibility in supporting a wide range of user devices and connectivity modes, not just mobile cellular users. Building on the foundation of MEC, VEC emerges as a specialized extension tailored for the dynamic environment of vehicular networks. While MEC was originally introduced to bring cloud

capabilities closer to mobile users, VEC adapts this concept to meet the unique demands of vehicles on the move.

One of the key distinctions between VEC and MEC lies in the characteristics and implications of mobility. In VEC, vehicles move at high speeds along structured routes, leading to rapid changes in network topology and frequent handoffs between edge nodes such as RSUs or other vehicles. This introduces challenges like intermittent connectivity, reduced offloading windows, and the need for ultra-low latency responses, especially for safety-critical applications. While MEC also deals with mobility, the movement is generally slower and less disruptive to network stability, resulting in longer-lasting connections and more predictable task execution. Although both paradigms may support applications with strict real-time requirements, the high-speed, large-scale, and coordinated mobility in VEC environments amplifies the difficulty of maintaining consistent performance. Consequently, VEC demands more adaptive, location-aware, and resilient task offloading strategies to cope with its uniquely dynamic conditions. In VEC, vehicles can offload tasks such as environment sensing, object detection, route planning, and traffic analysis to nearby edge servers located in RSUs or even other vehicles. This reduces the processing burden on individual vehicles, lowers energy consumption, and ensures fast response times, which are essential for safety and efficiency in transportation systems [51]. VEC is particularly valuable due to the high mobility of vehicles, the need for rapid decision-making, and the massive amount of data generated on the move. By utilizing local edge resources, VEC supports critical applications such as collision avoidance and intelligent traffic control, making it a vital component of modern vehicular networks [57].

Task offloading in vehicular networks involves transferring computationally intensive tasks from vehicles to more powerful external resources, such as edge servers located in RSUs or even other nearby vehicles. Modern vehicles are equipped with various sensors and systems that generate large volumes of data for tasks like real-time navigation, object detection, video processing, and collision avoidance. Processing all of this data locally can strain a vehicle's onboard computing resources and drain energy. To address this, task offloading allows vehicles to delegate demanding operations to edge servers through VEC, enabling faster processing and reducing the load on local systems [36]. This approach offers several advantages. It minimizes latency, which is critical for time-sensitive decisions, and improves overall system efficiency by balancing workloads across the network. Task offloading also enhances scalability, as it allows the network to handle increasing demands from a growing number of connected vehicles. However, successful offloading depends on factors like network connectivity, vehicle mobility, task deadlines, and server availability. As a result, designing intelligent offloading strategies is essential to ensure reliable performance, low latency, and effective use of available edge resources in vehicular environments [2].

In addition to task offloading, task partitioning can also be an effective approach in vehicular networks. Instead of offloading an entire task, task partitioning involves dividing it into smaller components, with certain parts executed locally on the vehicle and others processed by VEC servers. This method offers a balanced solution to challenges such as latency, energy consumption, and limited onboard computing power. By handling urgent or lightweight operations locally, such as obstacle detection or basic sensor processing, and offloading more complex or time-flexible components like image recognition or route planning, vehicles can maintain real-time responsiveness while still benefiting from edge resources. Task partitioning is particularly valuable when network conditions are unstable or when full offloading is not feasible. To implement it effectively, intelligent partitioning strategies are needed that take into account task requirements, current network performance, and available computing resources both locally and at the edge [10].

Moving on to the algorithms used in task offloading, deterministic approaches enable fast and straightforward decision-making based on predefined rules. Two commonly used methods are FCFS and SDF. FCFS processes tasks in the exact order they arrive, without considering task size or urgency, making it easy to implement but potentially inefficient under heavy load. In contrast, SDF prioritizes tasks with the closest deadlines, aiming to reduce the number of dropped tasks by handling the most time-sensitive ones first. While these methods offer fast and straightforward decision-making, they are not always reliable in complex and dynamic vehicular environments. FCFS processes tasks based solely on arrival time, ignoring factors like task size, urgency, or system load, which can lead to inefficient use of resources and increased latency. SDF improves on this by prioritizing tasks with the nearest deadlines, but it still overlooks other important aspects such as computation time, communication delays, and current server load [61].

To efficiently manage task execution in vehicular networks, optimization techniques play a critical role in determining the most effective way to process and allocate tasks. The main objective is to minimize the total latency and reduce the number of dropped tasks, especially in environments with limited computational resources and fluctuating network conditions. Total latency in this context is defined as the sum of communication and computation latencies. Communication latency includes the time taken for uplink and downlink transmissions. Computation latency, on the other hand, refers to the time it takes to process the task either locally on the vehicle or remotely on the edge server. Optimization is particularly valuable when dealing with task offloading and partitioning decisions. It helps in selecting not only where and how each task should be executed, whether locally, fully offloaded, or partially offloaded, but also in what order tasks should be scheduled to minimize delays and avoid overloading the system. Given the complexity and dynamic nature of vehicular environments, heuristic optimization algorithms are often

used due to their ability to find near-optimal solutions in reasonable time frames [103]. One such technique employed in this thesis is PSO, a population-based metaheuristic inspired by the social behavior of birds flocking or fish schooling. In PSO, a group of particles moves through the search space to find the optimal solution by updating their positions based on their own experience and the experiences of neighboring particles. Each particle adjusts its trajectory according to its best-found position and the best-known position among all particles, gradually converging toward an optimal or near-optimal solution. PSO is well-suited for task offloading and scheduling problems because it can efficiently handle the multi-dimensional search space created by numerous variables such as task sizes, processing deadlines, edge server loads, communication delays, and energy constraints. In the context of this thesis, PSO is used to find the best offloading and partitioning decisions for a set of tasks, ensuring that the overall system latency is minimized and that as few tasks as possible are dropped due to timeouts or resource shortages. By integrating PSO with a well-defined model of task execution and communication costs, the system can adapt to changing network conditions and dynamically optimize task handling in real time, improving the reliability and performance of VEC systems [88].

In highly dynamic environments like vehicular networks, static or pre-defined task offloading strategies often fall short due to rapidly changing conditions such as vehicle mobility, fluctuating network bandwidth, variable edge server loads, and unpredictable task arrival rates. These conditions demand a more responsive and adaptive approach known as dynamic task offloading. Dynamic task offloading allows vehicles to make real-time decisions on whether to process tasks locally, offload them entirely, or partition them, based on the current state of the system. This flexibility is crucial for maintaining low latency, ensuring reliability, and minimizing the number of dropped tasks in such time-sensitive scenarios [31].

While optimization techniques like PSO can provide effective offloading strategies by exploring a wide range of possible solutions, they are often not suitable for real-time decision-making in vehicular networks. PSO and similar metaheuristic methods typically require multiple iterations and considerable computation time to converge to a near-optimal solution. In fast-changing environments where task decisions must be made within milliseconds, this high execution time can lead to delays, rendering these methods impractical. Therefore, lightweight, fast, and context-aware offloading mechanisms are needed to support dynamic task offloading in real-world vehicular systems. These approaches prioritize quick adaptation over perfect optimization, aiming to maintain system responsiveness and service continuity in constantly evolving conditions [3]. ML has become a valuable approach for enabling intelligent decision-making in dynamic and complex environments. In the context of vehicular networks, ML techniques can learn patterns from data and adapt to

new situations, making them highly suitable for dynamic task offloading. Among various ML approaches, RL stands out as a promising solution for real-time decision-making in uncertain environments. RL allows an agent to learn the best offloading strategy through interaction with its environment. It does so by receiving feedback in the form of rewards or penalties based on its actions, gradually improving its decisions over time. This makes RL particularly effective in dynamic scenarios where conditions such as network bandwidth, server load, and vehicle mobility are constantly changing [47]. Unlike deterministic or static optimization methods, RL does not require prior knowledge of the environment or hand-crafted rules. Instead, it can continuously adapt its behavior based on current observations, aiming to minimize total latency and reduce the number of dropped tasks. With its ability to balance exploration and exploitation, RL can find near-optimal task offloading policies that respond efficiently to real-time system changes. This makes it a strong candidate for achieving fast, adaptive, and intelligent task management in VEC [66].

One widely used RL algorithm is DQN. It combines traditional Q-learning with Deep Neural Network (DNN) to approximate the Q-values, which represent the expected cumulative reward of taking a specific action in a given state. In the context of vehicular task offloading, DQN helps the agent decide whether to process a task locally, offload it entirely, or partially, by evaluating the long-term impact of each action. DQN is effective because it learns directly from experience and can handle high-dimensional state spaces, such as those involving multiple tasks, network conditions, and server loads. PPO extends the RL framework by directly optimizing the policy through policy gradient methods, offering a structured approach to learning optimal actions without relying solely on Q-value estimation. It employs a clipped objective function to stabilize policy updates, preventing excessive shifts that could disrupt learning. This stability is particularly advantageous in scenarios with continuous or complex action spaces, such as deciding optimal task offloading percentages or dynamically allocating resources in vehicular systems. PPO's structured update mechanism makes it a compelling choice for maintaining consistent performance in rapidly changing environments. Both DQN and PPO provide powerful tools for intelligent task offloading. DQN is simpler and effective for discrete action spaces, while PPO offers more flexibility and robustness in complex, dynamic scenarios. Their ability to learn and adapt to real-time conditions makes them highly suitable for optimizing task management in VEC systems [45].

1.1 Contributions

This thesis introduces a comprehensive framework for efficient task offloading in vehicular networks under realistic and high-stress conditions. The main contributions are as follows:

1. A theoretical performance benchmark is established using PSO in a static and offline setting, providing a reference for evaluating the performance of dynamic algorithms under ideal conditions.

2. A dynamic scheduling environment is developed using a decision window approach that intelligently handles incoming tasks before MEC availability, enabling timely and efficient task allocation while minimizing E2E latency and task drop rates.

3. A novel and fast cost-driven algorithm is proposed for real-time task scheduling, effectively managing continuous task arrivals and fluctuating resource availability with minimal computational overhead. Additionally, the system integrates pretrained RL models such as DQN and PPO, employing a dynamic reward function to adapt to varying scenarios without the need for retraining. In straightforward conditions, the system bypasses unnecessary intelligent decision-making to further reduce execution time while maintaining performance.

1.2 Publications

1. Mahsa Paknejad, Parisa Fard Moshiri, Murat Simsek, Burak Kantarci, and Hussein T. Mouftah. A reliable and efficient 5g vehicular mec: Guaranteed task completion with minimal latency. *In IEEE International Conference on Communications 2025*, March 2025.

2. Mahsa Paknejad, Parisa Fard Moshiri, Murat Simsek, Burak Kantarci, and Hussein T. Mouftah. On-dyn-cda: A real-time cost-driven task offloading algorithm for vehicular networks with reduced latency and task loss. *under review at IEEE Internet of Things Journal*, 2025.

3. Mahsa Paknejad, Parisa Fard Moshiri, Murat Simsek, Burak Kantarci, and Hussein T. Mouftah. Meeting deadlines in motion: Deep rl for real-time task offloading in vehicular edge networks. *In International Conference on Network of the Future 2025*, July 2025.

The next chapter offers an overview of existing research relevant to this field. Chapter 3 details our investigation into static task offloading integrated with task partitioning through the use of PSO. In Chapter 4, we present an innovative dynamic algorithm designed for real-time systems, which achieves notable reductions in execution time and surpasses PSO in dynamic scenarios. Chapter 5 delves into the use of RL methods, focusing on DQN and

PPO, to enable intelligent and adaptive task offloading. Finally, the last chapter wraps up the thesis.

Chapter 2

Literature Review

2.1 Optimization and Performance Evaluation in VEC

One of the central challenges in task offloading within VEC lies in the highly dynamic and time-constrained nature of vehicle connectivity. Vehicles typically remain within the coverage area of an RSU for only a brief period. During this short time, they must offload and receive responses for computation-intensive tasks. However, when multiple vehicles attempt to offload tasks simultaneously, RSU can quickly become overwhelmed, resulting in long queues of pending tasks. Many researchers explore solutions to tackle these challenges. One promising approach is the use of optimization techniques that manage task offloading decisions to balance workloads and minimize latency. One study focuses on optimizing the E2E latency through improved decision-making policies for assigning tasks to MEC hosts. Three task assignment strategies are evaluated: a static pre-defined mapping, a First-Fit approach based on service availability, and a Best-Fit strategy that considers CPU resource availability across MEC nodes. This work highlights the limitations of static orchestration policies and advocates for future development of dynamic, mobility-aware algorithms [76].

Continuing the discussion on static optimization, a low-latency scheduling approach tailored for dependent tasks in MEC-enabled 5G vehicular networks is proposed. Unlike traditional methods that focus on independent tasks, their work addresses the more realistic scenario of interdependent subtasks. They introduce a Priority-Based Task Scheduling Algorithm (PBTSA), which first models tasks as a Directed Acyclic Graph (DAG), calculates subtask priorities using a Reverse Breadth-First Search (RBFS), and then performs greedy offloading to minimize overall processing delay [97]. Besides binary offloading, which assumes that each task is indivisible and must be executed either entirely locally or entirely

at the edge, one approach is partial offloading, where a task can be split such that a portion is processed locally while the remainder is offloaded to the edge server. In this context, a study explores both schemes within MEC-enabled vehicular networks, formulating latency minimization problems for each. They develop efficient algorithms by decomposing the joint computation offloading and resource allocation problems into tractable subproblems using convex optimization techniques. Their results show that partial offloading, due to its flexibility in distributing computational workload, offers superior latency performance compared to binary offloading [44].

Moreover, a joint optimization strategy that supports both full and partial task offloading based on current network conditions is proposed. Vehicles with limited computing capacity and energy are considered. Their method jointly optimizes the offloading ratio and transmit power, and introduces a channel gain threshold to determine whether full or partial offloading is more efficient. They transform the original non-convex problem into two convex sub-problems and solve them using conventional optimization techniques [75].

In a similar work, authors propose a joint optimization strategy that considers both user mobility and inter-user task relevance, allowing for partial task offloading. They formulate the problem as a mixed-integer program and solve it using an iterative algorithm based on the one-time offloading principle. They also incorporate a mobility-aware delay constraint to ensure reliable task execution. Simulation results show that their approach significantly outperforms full local or full offloading strategies in terms of efficiency [37].

Another study on static optimization proposes a two-stage collaborative task offloading strategy using static, pre-collected task and network information. In the first stage, they use a task triage method based on urgency and criticality to make quick offloading decisions for tasks with extreme characteristics. In the second stage, they model the remaining task offloading problem as a multi-objective optimization problem and solve it using Multi-Objective Hybrid Genetic Algorithm (MOHGA) enhanced by a Bayesian Maximum Entropy (BME) framework [11].

One of the most well-known swarm intelligence algorithms widely used for solving complex optimization problems is PSO. Due to its simplicity and effectiveness in handling nonlinear and multidimensional problems, PSO is applied in various domains such as task scheduling, resource management, and network optimization. Building on this, a PSO-based algorithm called particle swarm optimization based task offloading and resource allocation (PTORA) is proposed to jointly optimize offloading decisions and the allocation of computing and communication resources. By considering the heterogeneity of edge resources and introducing a linear decrement strategy to enhance search performance, PTORA achieves better results compared to traditional methods [56]. Following the previous work, a joint task offloading and resource allocation framework in a Software-Defined

Networking (SDN)-enabled VEC environment is deployed. They design a multi-objective optimization model to minimize both processing delay and cost, considering the limited computational resources of vehicles and edge servers. To solve this complex nonlinear problem, they develop a PSO-based algorithm called Particle swarm optimization-based Task offloading and Resource allocation (PTR) that searches for Pareto-optimal solutions [100].

One key aspect of task offloading is the communication component. Since many vehicles may attempt to connect to the same server simultaneously, efficient utilization of communication resources becomes essential. Non-Orthogonal Multiple Access (NOMA) enables multiple vehicles to transmit data concurrently over the same frequency band. By applying a decoding method known as Successive Interference Cancellation (SIC), the server can separate and process these overlapping signals, enhancing spectrum efficiency and reducing task offloading delay. In a recent study, researchers propose an analytical model for NOMA-assisted MEC in vehicular networks that considers vehicle mobility, including speed and arrival rate. They derive expressions for average task offloading latency and task loss probability, and validate their model through simulations. The results show that task loss probability is primarily influenced by vehicle arrival rate and is largely unaffected by average vehicle speed. This work emphasizes the need to account for mobility when developing task offloading strategies in NOMA-based MEC environments [41].

One line of study in task offloading explores the use of nearby assistant vehicles to help distribute computational tasks. Building on this idea, researches proposes a joint Vehicle-to- Infrastructure (V2I)–Vehicle-to-Vehicle (V2V) offloading framework where autonomous vehicles can offload tasks either to an RSU via V2I communication or to idle neighboring vehicles via V2V links. This approach aims to minimize overall task latency by balancing the computational load across available edge and vehicular resources. The system intelligently selects the offloading destination based on factors such as channel conditions, vehicle density, and resource availability, while also accounting for communication impairments like Doppler spread due to high vehicle mobility [112].

Given the high mobility of vehicular networks, it is important to support dynamic decision-making to ensure practical and efficient task execution. Building on this, a recent study proposes Bee colony-based Task offloading in Vehicular edge computing (BTV), which uses the Bee Colony algorithm to perform dynamic task scheduling and offloading. Similar to PSO, the Bee Colony algorithm is a population-based metaheuristic inspired by natural swarm intelligence. While PSO models the social behavior of bird flocks or fish schools, the Bee Colony algorithm simulates the foraging behavior of honey bees to explore and exploit the solution space. By incorporating contextual information such as CPU availability, energy levels, communication quality, and task size, BTV aims to minimize execution time and ensure reliability in real-time vehicular environments [18].

Game theory is often used in task offloading to model the interactions among multiple vehicles or devices that compete for limited edge computing resources. Each player (vehicle) aims to maximize its own benefit, such as reducing delay or saving energy, by choosing whether to offload a task or compute it locally. Game theory helps find a stable solution where no player can improve its outcome by changing its decision alone; this is known as a Nash Equilibrium. In a recent paper, the authors apply game theory to design a distributed task offloading strategy. Each vehicle independently decides whether to offload its task or process it locally, based on its own delay and energy constraints. They prove that the offloading game reaches a Nash Equilibrium and integrate this with resource allocation using the Lagrange multiplier method. This forms the core of their proposed Mobility-Aware Computational Efficiency-Based Task Offloading and Resource Allocation (MACTER) algorithm, which improves computation efficiency in dynamic vehicular networks [74].

Continuing with dynamic optimization, a Lyapunov-Based Profit Maximization (LBPM) algorithm is deployed for independent task offloading in MEC-enabled 5G Internet of Vehicles (IoV). Their method dynamically makes offloading decisions in each time slot using the drift-plus-penalty framework of Lyapunov optimization. The goal is to maximize the time-averaged profit of MEC providers while ensuring service-level agreements are met. Tasks are assigned to MEC servers through a matching process solved by the Hungarian algorithm. The approach supports variable task lengths and non-preemptive execution, adapting to real-time system changes [82].

In task offloading for VEC, Mixed-Integer Nonlinear Programming (MINLP) is often used to model complex decision-making problems. This is because offloading decisions involve both discrete variables, such as whether to offload a task or which server to choose, and continuous variables, such as CPU frequency and transmission power. MINLP allows the combination of these decision types within a single optimization framework, but it is also difficult to solve due to its non-convex and computationally intensive nature. In a recent study, the authors use an MINLP model to jointly optimize task offloading and service caching in a VEC system. They consider that each application consists of multiple subtasks with dependencies, represented by a directed acyclic graph, and that edge servers can cache certain executable services to improve efficiency. To manage the complexity of the MINLP problem, the authors propose a semi-distributed algorithm called Dependency-Aware Task Offloading and Service Caching (DTOSC). In this approach, each vehicle first optimizes its local resource usage such as CPU frequency and transmission power and calculates task priorities. Then, a centralized SDN controller uses this information to make task offloading, scheduling, and caching decisions [78]. In task offloading, VEC-VEC offloading refers to the horizontal transfer of tasks between nearby VEC servers. This

is useful when one VEC server is overloaded while another nearby has spare resources. VEC-cloud offloading, on the other hand, involves the vertical transfer of tasks from VEC servers to the remote cloud. This is typically used when local edge resources are insufficient to handle the computational load or when specific tasks require more powerful processing capabilities. As a recent work, both VEC-VEC and VEC-cloud offloading are supported through a cooperative framework. The authors design a hierarchical system where an SDN controller coordinates resource allocation and offloading decisions. They propose the BARGAIN-MATCH algorithm, which enables efficient task migration across edge servers (VEC-VEC) and to the cloud (VEC-cloud) by using game theory and matching strategies. This approach helps balance the workload, reduce delays, and improve overall system performance [85].

Resource-aware decision-making in task offloading focuses on selecting where to offload tasks based on the current availability of computing resources and network conditions. It ensures that each task is handled by a node, either local or at the edge, that can meet its latency requirements effectively. This approach often considers the dynamic state of the system, such as server load, communication delay, and vehicle location, to make real-time decisions that improve system performance. Following this idea, the paper applies resource-aware decision-making by jointly optimizing task offloading and resource allocation through a priority-aware strategy. It introduces a variant grey wolf optimizer (VGWO) to efficiently allocate MEC server resources and employs a Dynamic Task Offloading Strategy (DOS) to determine where each task should be executed. The proposed approach prioritizes tasks with stricter delay requirements and adapts offloading decisions to the current resource availability across the network [96].

Greedy models in task offloading make quick, local decisions by selecting the best immediate option, such as offloading a task to the server with the shortest processing time or lowest current load. These methods are efficient and simple to implement but may lead to suboptimal global performance since they ignore the long-term impact of decisions. Building on this, a proposed approach first applies a greedy strategy based on the Weighted Shortest Processing Time (WSPT) rule to generate an initial offloading plan. Tasks with higher delay sensitivity and shorter execution times are given priority. This initial solution is then refined using a combination of simulated annealing and heuristic rules. The simulated annealing process helps escape local optima by accepting worse solutions with a certain probability, while heuristics guide the selection of tasks and servers based on their cost impact. The method jointly optimizes both task placement and execution order, aiming to minimize delay and operational costs [32].

In contrast to static greedy approaches, a probabilistic task offloading model is designed specifically for high-mobility vehicular networks. Their method introduces a three-tier

architecture consisting of Vehicle Edge of Things (VEoTC), VEC, and Cloud Computing (CC) to dynamically offload tasks based on connectivity, resource availability, and task priority. Service Vehicles (SVs) equipped with computing resources act as mobile edge nodes, allowing longer and more stable connections compared to stationary RSU. They model the system using queueing theory and calculate offloading probabilities and E2E delays for each computing layer. Real mobility data is used to estimate dwell time and validate the advantages of mobile offloading, showing improved latency performance and reduced handover penalties [35].

Building on the previous work, a recent paper proposes a mobility-aware multi-hop task offloading scheme for VEC. Their approach leverages both one-hop and multi-hop neighboring vehicles to maximize task execution opportunities in highly dynamic vehicular environments. They model the offloading problem as a utility minimization problem balancing execution delay and economic cost. A semidefinite relaxation method coupled with an adaptive adjustment algorithm is used to solve the optimization problem under mobility and connectivity constraints. Simulation results show significant delay reductions compared to baseline approaches, demonstrating the efficiency of utilizing underutilized vehicle resources for edge computing tasks [48].

Another recent work investigates a Cybertwin-enabled multi-vehicle task offloading architecture for MEC in 6G networks. This approach considers a hybrid energy-powered MEC system where vehicles and edge servers share real-time status and task information with macro Base Stations (BSs) to facilitate efficient resource allocation. The authors formulate a stochastic optimization problem to minimize overall system cost. They decompose the problem into sub-problems and propose an Multi-Vehicle Task Offloading (EMT) algorithm based on Lyapunov optimization. This algorithm enables dynamic, real-time task scheduling without requiring prior statistical information [13].

Building on this, a comprehensive framework named Partial computation Offloading and adaptive Task Scheduling algorithm (POETS) is introduced for partial computation offloading and adaptive task scheduling in 5G-enabled vehicular networks. In this approach, vehicles offload part of their computation tasks to nearby MEC servers using NOMA-based communication. The framework jointly optimizes channel allocation, offloading ratio, and payoff strategies to maximize system-wide profit while ensuring individual rationality and incentive compatibility. A two-sided matching algorithm is used for channel assignment, convex optimization determines the optimal offloading ratio, and a non-cooperative game models the payoff policy. Simulations with real-world taxi traces demonstrate notable improvements in delay reduction and overall efficiency [68].

Similarly, an Adaptive Data Rate-based Offloading algorithm (ADRO) is proposed for vehicular networks, which accounts for varying V2V communication rates due to vehicle

mobility. The algorithm ensures that tasks meet their time constraints by selecting nearby vehicles as computation resources when possible and allowing tasks to wait for resource availability when necessary. Offloading decisions are prioritized based on the availability of offloading vehicles and the urgency of the task. Simulation results based on real-world vehicle traces show that the algorithm improves task scheduling efficiency compared to baseline approaches [9].

These key contributions are outlined in Table 2.1, offering a comprehensive gap analysis that contrasts this thesis with recent studies.

Table 2.1: Assessment of our work in relation to existing research

Paper	**Theoretical Limit**	**Offloading Mode**	**MTG**	**Training**		**KPI**			**Algorithm**
				Offline	**Online**	**AET**	**Latency**	**DTN**	
[76]	✗	Static	Simu5G, OMNET++	-	-	✗	✓	✗	pre-defined association, First-Fit, Best-Fit
[97]	✗	Static	Python	-	-	✓	✓	✗	PBTSA, Probability-Based Location Aware (PBLA), TBTOA
[44]	✓	Static	-	-	-	✗	✓	✗	Communication Resource Allocation, Partial/Binary Offloading
[75]	✓	Static	-	-	-	✗	✓	✗	Full/Partial Offloading
[37]	✗	Static	-	-	-	✗	✓	✗	Combined iterative algorithm based on the one-time offloading, all offloading computing, all local computing

(Table 2.1 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[11]	✗	Static	MATLAB	-	-	✓	✓	✗	MOHGA, Random Weighting Genetic Algorithm (RWGA), Grey Entropy Parallel Analysis (GEPAGA), Fuzzy Relative Entropy Genetic Algorithm (FREGA), Non-ominated Sorting Genetic Algorithm (NSGA)-III, Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D), strength Pareto-based evolutionary algorithm (SPEA)-II
[56]	✗	Static	c++	-	-	✗	✓	✗	PTORA, All Edge Computing (AEC), All Local Computing (ALC)
[100]	✗	Static	-	-	-	✗	✓	✗	PTR, AEC, ALC
[41]	✓	Static	-	-	-	✗	✓	✓	Analytical NOMA assisted MEC model

(Table 2.1 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[112]	✗	Static	-	-	-	✗	✓	✗	MEC for Vehicle-to-Everything (MEC-V2X), Local-fixed, Pure-MEC, MEC-Local, V2Vrandom-fixed, V2V-optimal-fixed, V2V-optimal-exhausted
[18]	✗	Dynamic	Simulation of Urban Mobility (SUMO), ns-3	-	-	✓	✓	✓	BTV, First In, First Out (FIFO), Multidecision Based Offloading (MDO), greedy task by task (GTT)
[74]	✗	Dynamic	MATLAB	-	-	✗	✓	✗	Resource Allocation and Offloading Strategy, distributed MACTER
[82]	✗	Dynamic	Python	-	-	✓	✓	✓	LBPM, CGTAS, MGW, MSCPC

(Table 2.1 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[78]	✗	Dynamic	-	-	-	✗	✓	✓	DTOSC, Priority and Dependency-Based DAG Tasks Offloading (PDAGTO), Task Offloading based on Ant Colony Optimization (TOACO), Task Offloading without Service Cache (TOWSC), Task Offloading without Task Urgency (TOWTU)
[85]	✗	Dynamic	MATLAB	-	-	✓	✓	✓	BARGAIN-MATCH
[96]	✗	Dynamic	-	-	-	✗	✓	✗	VGWO, DOS, Cuckoo Search (CS), Whale Optimization Algorithm (WOA), Grey Wolf Optimizer (GWO), Local Offloading Strategy (LOS), Random MEC Server for Offloading Strategy (RMOS)

(Table 2.1 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[32]	✗	Dynamic	SUMO, Veins	-	-	✗	✓	✗	Joint Task Offloading and Scheduling Algorithm (JTOS), Nearby offloading (NO), Node select optimization (NSO)
[35]	✗	Dynamic	Jupyter, MATLAB	-	-	✗	✓	✓	three-tier dynamic offloading across VEoTC, VEC, and Cloud
[48]	✗	Dynamic	-	-	-	✗	✓	✗	Mobility-Aware Multi-hop Task Offloading (MMTO), Local Processing Algorithm (APA), Random Policy Algorithm (RPA), MMTO-O, RPA-O
[13]	✗	Dynamic	MATLAB	-	-	✗	✓	✗	EMT, Lyapunov Optimization-Based Dynamic Computation Offloading (LODCO), Random Allocation Strategy (RAS), Equal Allocation Strategy (EAS)

(Table 2.1 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[68]	✗	Dynamic	Python	-	-	✗	✓	✗	POETS, DTMS, IHRA, MECO, POETS without NOMA access (POETS w.o. NOMA)
[9]	✗	Dynamic	SUMO	-	-	✗	✓	✗	ADRO, random policy, greedy policy
Ours	✓	✓	SUMO	✓	-	✓	✓	✓	FCFS, SDF, Off-Sta-PSO, On-Sta-PSO, Online Dynamic PSO (On-Dyn-PSO), On-Dyn-CDA, DQN, PPO

** MTG: Mobility Trace Generator, KPI: Key Performance Indicator, AET: Algorithm Execution Time, DTR: Dropped Tasks Ratio*

2.2 ML and RL Approaches in VEC

ML has been widely adopted in MEC to enhance decision-making by enabling systems to learn patterns from data and adapt to changing conditions. Among various ML approaches, RL stands out as particularly effective for task offloading in dynamic environments. RL learns optimal policies through interaction with the environment, making it well-suited for VEC where network conditions and resource availability frequently change. By modeling the offloading problem as a sequential decision-making task, RL can balance trade-offs between delay and service reliability. Building on advantages of RL, a joint offloading strategy is introduced for MEC-enabled vehicular networks that utilizes Federated Reinforcement Learning (FRL) and Adaptive Particle Swarm Optimization (APSO) to minimize task delay. The method selects between local computing, V2V offloading to nearby vehicles, or MEC-based RSU offloading based on task requirements and resource availability. An adaptive PSO algorithm with crossover and shrinkage-expansion enhancements improves convergence and search efficiency. Simulation results demonstrate that this approach significantly reduces service time compared to existing methods such as

FCFS, RMS, and fuzzy-based strategies [80]. Extending the prior work, a recent study introduces a fully distributed task offloading (FDTO) framework for VEC, in which each vehicle autonomously determines its offloading strategy using only local information from nearby RSUs. This method tackles key challenges including rapid vehicle movement, constrained RSU resources, and the scalability demands of large vehicular networks. The authors develop two vehicle-side algorithms: a simple greedy method (FDTO-VG) and a convex optimization-based approach (FDTO-VX), and demonstrate that both converge to the global optimum [53].

Furthermore, a dynamic offloading framework based on DRL is proposed for highly mobile VEC environments. The method formulates task offloading as a cost minimization problem, incorporating both delay and energy usage. Leveraging a Double Deep Q-network (DDQN), the system learns optimal offloading decisions in real time, even under uncertain and changing network conditions. A handover-enabled mechanism allows vehicles to update their offloading strategy during movement, improving quality of service. Simulation results show that this approach significantly outperforms traditional and existing RL-based offloading strategies in terms of cost efficiency and adaptability [54].

Building on the effectiveness of deep RL for VEC, a graph-powered task offloading framework is proposed that integrates Graph Convolutional Networks (GCNs) with DQNs. The framework models the vehicular network as a graph and uses GCNs to enhance state representation by capturing the structural and feature information of nodes. Task offloading is formulated as a Markov Decision Process, allowing the agent to learn optimal server selection policies across vehicles, edge servers, and cloud nodes. The reward function considers delay constraints, offloading costs, and resource utilization. Simulation results show that the proposed scheme significantly improves task acceptance rates, reduces task rejection, and achieves higher cost efficiency and resource utilization compared to heuristic methods [87].

Extending this line of research, a DRL-based offloading framework called COADRL is introduced to address stochastic task arrivals and time-varying channel conditions in heterogeneous vehicular networks. Instead of relying on prior knowledge, the framework employs a decentralized learning approach using Double DQNs with experience replay to adaptively determine optimal task migration and resource allocation strategies. The model incorporates channel state, execution delay, and bandwidth cost into its state and reward formulations, enabling more effective decision-making under real-world constraints. Simulation results confirm that COADRL achieves higher cumulative rewards and throughput while reducing total system costs compared to several benchmark algorithms [5].

Following this, Mobility-aware dependent task offloading (MESON) is proposed as a mobility-aware dependent task offloading scheme for urban VEC. It introduces a DRL-

based framework utilizing the Deep Deterministic Policy Gradient (DDPG) algorithm to optimize offloading decisions in the presence of task dependencies and dynamic vehicle trajectories. To enhance decision accuracy and reduce action space complexity, MESON integrates a mobility detection algorithm that predicts communication durations between vehicles and RSUs. It also employs a task priority determination mechanism based on multiple criteria to improve task scheduling efficiency [114].

Moreover, a RSU-assisted learning-based partial task offloading algorithm is proposed to further address the challenges of dynamic network conditions in VEC. It enhances adaptability by allowing vehicles to share learned information via RSUs, improving decision-making in highly dynamic environments. Additionally, it supports partial task offloading for parallel execution, further reducing task delay and improving performance compared to traditional methods [43].

Furthermore, building upon the idea of learning-based adaptation to dynamic environments, this work adopts a Meta Reinforcement Learning (Meta-RL) framework to further enhance generalization across diverse offloading scenarios. While the RSU-assisted approach leverages shared learning for improved decision-making, this method formulates the offloading problem as a sequence-to-sequence task and utilizes a Bi-GRU with attention to capture subtask dependencies. A model-agnostic meta-learning strategy enables rapid adaptation to new task and network conditions with minimal retraining. This approach achieves lower execution delays and improved scalability in multi-task VEC [16].

Building on this, a fusion of model-driven and data-driven intelligence is introduced to further enhance offloading efficiency in dynamic urban vehicular environments. This method constructs detailed models for task queues, communication, computation, and cost, enabling precise representation of system dynamics. By formulating the offloading problem as a Markov Decision Process, it employs a multi-agent proximal policy optimization algorithm with a centralized training and decentralized execution framework. Convolutional neural networks are used to extract relevant features from complex state information, allowing agents to make informed task and target selection decisions. This approach achieves improved task completion latency and system utility, complementing meta-learning strategies by providing robust performance in large-scale, real-time settings [34].

In another work, researchers propose a distributed DRL approach that further enhances offloading efficiency by explicitly addressing subtask dependencies in vehicular edge networks. Their method models task structures as directed acyclic graphs (DAGs) and introduces a priority-based scheduling mechanism to preserve execution order. To tackle privacy constraints and reduce reliance on global information, they formulate the offloading problem as a partially observable Markov decision process and design a Optimized Distributed Computation Offloading (ODCO) scheme. The use of Convolutional Neural

Network (CNN)s and transformers enables each vehicle to make effective decisions based on local and historical information. This approach complements prior work by enabling dependency-aware, privacy-preserving offloading decisions with improved system utility and convergence speed in multi-agent scenarios [46].

Extending the previous work on dependency-aware and privacy-preserving offloading, another study introduces a multilayer DRL framework tailored for cloud–edge–end collaborative offloading in vehicular edge networks. This approach incorporates a realistic queue-based model to manage multislot tasks and addresses the joint optimization of offloading, scheduling, and resource allocation. By decoupling decisions across three layers: offloading (via D3QN), scheduling with transmission power control (via PDQN), and resource allocation (via convex optimization), it enables vehicles to make asynchronous, online decisions that respond to dynamic network states. This complements earlier efforts by enhancing scalability and responsiveness in complex multi-agent environments [99].

Building on the previous efforts, a recent study proposes a Generative Adversarial Network (GAN)-based federated optimization framework named Mobile Edge Computing Assisted Task Offloading using Generative Adversarial Network (MEGAN) to address vehicular task offloading challenges in 5G/6G-enabled MEC. It integrates task representation learning with federated distillation learning to optimize task classification, energy consumption, and system throughput while preserving user data privacy. GAN is employed to generate balanced and diverse task datasets, mitigating overfitting and data imbalance. The framework uses semi-supervised learning and federated model updates between edge servers and vehicles for efficient and privacy-preserving task processing. Experimental results demonstrate that MEGAN outperforms existing methods in classification accuracy, energy efficiency, and failure rate under various task loads and network stress conditions [98].

Another recent study applies DRL to the task offloading problem in multi-access edge computing environments. The authors design and compare tabular Q-learning and deep Q-learning methods to optimize task offloading decisions based on parameters such as task size, complexity, and timeout. The deep learning-based approach uses neural networks and experience replay to manage large state spaces more effectively than tabular methods. Experimental results show that deep Q-learning achieves lower task drop rates, reduced latency, and improved energy efficiency, demonstrating its superiority in dynamic edge computing scenarios [52].

Building on the previous work, another study addresses the task offloading problem in multi-server multi-access edge vehicular networks by proposing a DDQN-based approach. Similar to the deep Q-learning method, this approach uses RL to handle large and dynamic state spaces, but extends it by incorporating a two-stage decision process. The first stage

makes offloading decisions based on mobility patterns and task priority, while the second stage schedules offloading requests using a DDQN algorithm to optimize server resource utilization. Experimental results confirm that this method achieves higher task completion for important tasks and maintains stable performance in highly dynamic environments, improving upon conventional DQN and heuristic methods [93].

Extending this line of research, a Meta-RL-based approach is introduced to further enhance adaptability and efficiency in dynamic VEC environments. Similar to previous methods that leverage RL for offloading decisions, this approach models applications as DAGs to capture task dependencies. It employs recurrent neural networks to learn and adapt offloading strategies across varying scenarios. By incorporating a meta-policy framework, it enables rapid policy updates and generalization across diverse tasks. Experimental results demonstrate improved performance in both latency and energy consumption, showing clear advantages over conventional algorithms such as Greedy and PPO [116].

Building on the prior work, authors in a related work propose a joint service caching and computation offloading scheme tailored for VEC systems. They formulate the problem as a MINLP model to minimize average task processing delay, considering dynamic task requests and constrained edge resources. To solve this complex problem efficiently, they develop a DRL approach based on the DDPG algorithm. This enables adaptive decision-making on caching and offloading strategies. Simulation results show that their method significantly reduces delay and energy consumption compared to existing benchmarks [102].

Another related work introduces a delay-aware task offloading scheme for VEC that leverages sojourn time-based routing. The authors model the offloading problem as a Markov Decision Process and propose an Asynchronous Advantage Actor-Critic (A3C)-based DRL algorithm to optimize offloading decisions across vehicle-to-vehicle and vehicle-to-infrastructure links. Their approach supports partial task offloading and accounts for vehicle mobility and communication constraints. Simulation results confirm that their method significantly reduces system latency compared to other benchmark algorithms [108].

A further line of research proposes a Distributed Intelligent Task Offloading and Workload Balance (DIOW) framework for large-scale VEC systems. The framework considers both task offloading decisions from task vehicles and workload balancing among BSs. It aims to minimize system delay while satisfying energy consumption constraints. To solve the optimization problems involved, the authors adopt a Multi-Agent Deep Deterministic Policy Gradien (MADDPG) algorithm. Simulation results show that their method effectively reduces system delay and energy deficit compared to baseline approaches [50].

A related approach introduces a Vehicular Edge–Cloud Computing (VECC) framework to handle delay-sensitive task offloading by utilizing both edge and cloud resources. The

authors formulate the offloading problem as a Markov Decision Process and develop two RL algorithms: Advantage-Oriented Dueling Actor-Insulator Network (AODAI), a value-based method using a dueling actor-insulator network, and Actor–Critic Task Offloading (ACTO-n), a policy-based actor–critic scheme. These methods aim to minimize the time that task execution exceeds its quality threshold. Experimental results on real-world bus trajectory data show that their approach outperforms existing RL-based and greedy strategies, reducing tolerance time by at least 8.81% compared to other RL algorithms [7].

As a continuation of this effort, a related study proposes a collaborative task offloading approach that integrates local processing, MEC, and Vehicular Cloud (VC) strategies. The authors construct a dynamic vehicular cloud by clustering nearby vehicles with idle computing resources using a K-means algorithm. They select a VC leader to manage task distribution among cluster members. To optimize offloading decisions, they formulate a delay and energy minimization problem and solve it using Nelder-Mead and Differential Evolution algorithms. This method allows tasks to be processed locally, offloaded to nearby MEC servers, or distributed within the VC, depending on current network and resource conditions [83].

Another method adopts a federated RL approach to optimize task offloading in edge computing environments. The authors model the problem as a Markov Decision Process and propose Federated Learning-Twin Delayed Deep Deterministic Policy Gradient (FL-TD3), a novel algorithm that integrates Federated Learning with the Twin Delayed Deep Deterministic Policy Gradient (TD3) framework. This method enables distributed training across edge nodes, improving convergence speed and model stability while handling high-dimensional state-action spaces. The experimental results show that FL-TD3 outperforms conventional RL methods like TD3 and DDPG, achieving lower latency and energy consumption in vehicular edge scenarios [12].

Building on this line of work, another study proposes an intelligent task offloading scheme that integrates an improved Soft Actor-Critic (SAC) algorithm to reduce latency in VEC environments. The approach leverages task divisibility to minimize interruption by distributing subtasks across multiple edge nodes based on vehicle mobility, location, and resource availability. To enhance the SAC algorithm, the method introduces a dual experience replay mechanism and a value-difference based exploration strategy, improving both sample efficiency and exploration capability. Results demonstrate significant latency reduction and faster convergence compared to baseline methods like Q-learning and DQN, thus extending prior efforts in RL-based offloading with improved robustness and responsiveness [101].

Continuing from the previous work, another approach addresses the challenges of non-stationary and heterogeneous VEC environments by formulating the task offloading prob-

lem using online and off-policy learning methods based on multi-armed bandit theory. This method focuses on dynamic network and BSs selection by predicting latency through historical offloading data, enabling vehicles to adapt to fluctuating traffic loads. It models network dynamics with a piece-wise stationary process and incorporates a change-point detection mechanism to optimize network selection over time. Furthermore, it includes a relaying mechanism based on sojourn time to reduce task loss due to vehicle mobility. These contributions extend RL-based offloading strategies by incorporating non-stationary adaptation, historical learning, and mobility-aware relaying mechanisms for improved latency reduction and task reliability [8].

Building upon the prior work, a recent approach proposes a multi-vehicle-assisted MEC system that leverages both vehicle-mounted and RSU-based computing resources. It applies an Actor-Critic DRL framework to dynamically decide task execution locations between vehicles and RSUs, aiming to minimize total system cost, which includes both delay and energy consumption. The method models the environment as a Markov Decision Process and uses continuous action spaces to improve flexibility and learning efficiency. Compared to traditional DQN approaches, this solution accelerates convergence, enhances task offloading performance, and achieves lower system costs, thereby extending learning-based offloading strategies with finer-grained decision-making and cooperative vehicle assistance [89].

Extending this work, another method introduces a frame-based dynamic offloading scheme tailored for sequential subtasks in vehicular environments. It addresses the limitations of slot-based models by enabling immediate decision-making upon subtask generation, thereby reducing unnecessary delays. This approach formulates offloading as a Markov Decision Process and applies a DDQN to capture the sequential dependencies between subtasks. By allowing offloading decisions to span multiple RSUs and leveraging fine-grained subtask segmentation, it enhances flexibility in resource allocation under varying vehicle mobility and congestion levels. Experimental results confirm that this method significantly improves delay, transmission time, and waiting time compared to both traditional greedy algorithms and standard DRL-based models [86].

Continuing from the previous work, another approach presents a cooperative computational offloading strategy using a model-based DNN tailored for MEC in vehicular networks. It enhances service reliability and reduces delay by enabling smart task division and parallel processing between multiple RSUs. This method offloads computational tasks from vehicles to RSUs, which then cooperate by sharing subtasks to be processed in parallel, depending on the vehicle's trajectory and network conditions. A DNN is trained to optimize decisions related to offloading, task division, processing, and routing, ensuring minimal service delay even under high mobility. Unlike previous DRL-based methods, this

model focuses on centralized, data-driven decision-making and utilizes intelligent routing to guarantee result delivery, thereby addressing mobility-induced failures and achieving superior performance in terms of delay and reliability [65].

Building on the previous work, a complementary approach introduces a distributed clustering-based cooperative VEC framework to support real-time task offloading. It addresses the challenges of limited computing resources and uncertain task requirements by dynamically grouping vehicles into cooperative clusters based on their available computation power, connection duration, and task deadline distributions. Each cluster operates as a mobile edge server, coordinated by a head vehicle selected through a contribution-based metric. The system formulates the offloading decision as a "Less than or Equal to" Generalized Assignment Problem (LEGAP), and solves it using both an offline bound-and-bound optimal algorithm and an online heuristic method. This approach effectively balances long-term optimization with real-time adaptability, and achieves improved service revenue and service ratio compared to existing online strategies, particularly under dynamic traffic and mobility conditions [92].

Extending the prior approach, a complementary method introduces a trust-aware VEC framework that incorporates both V2V and V2I communication modes to enhance the flexibility and security of task offloading. To cope with the risks posed by malicious nodes and dynamic vehicular environments, it employs a recommendation-based trust model that combines direct and indirect trust metrics for evaluating node credibility. Task offloading is then optimized using a DDPG RL algorithm, which jointly minimizes delay and maximizes trustworthiness. This method effectively adapts to environmental changes and improves overall offloading efficiency compared to traditional and game theory-based approaches [109].

As a continuation of these efforts, this thesis builds upon the foundational concepts established in previous research while addressing several important aspects that are often missing in the existing literature. Unlike prior works, which typically focus on a limited set of challenges, this thesis adopts a more comprehensive approach. It takes into account the number of dropped tasks to evaluate system reliability, compares the proposed model with a theoretical upper bound to assess how close it comes to optimal performance, and implements bandwidth sharing to represent realistic network behavior. In addition, the work covers both static and dynamic offloading strategies, making it suitable for a wider range of vehicular scenarios. It also evaluates optimization techniques in dynamic environments where mobility and communication conditions frequently change. To ensure practical applicability, the actual execution times of the proposed algorithms are reported. While some of these aspects have been studied individually in earlier research, not all prior works have addressed them together in a unified framework.

These distinctions and contributions are summarized in Table 2.2, which provides a detailed gap analysis comparing this study with recent approaches.

Table 2.2: Assessment of our work in relation to existing research

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[80]	✗	Dynamic	-	-	-	✗	✓	✗	FRL + APSO, DTA DP, Fuzzy, RMS, FCFS
[53]	✓	Dynamic	Python	-	-	✓	✓	✗	FDTO-VG, FDTO-VX, FDTO-S, FDTO-V, FDTO-VG, FDTO-VX, Centralized Task Offloading (CTO)-DO, CTO-EO, Noncooperative Game based Task Offloading (NGTO), Multi-armed Bandit (MAB)
[54]	✗	Dynamic	Python	-	✓	✗	✓	✗	DDQN, Adaptive Learning-Based Task Offloading (ALTO), Upper-Confidence-Bound (UCB), Adaptive Upper-Confidence-Bound (AdaUCB)

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[87]	✗	Dynamic	Python	-	✓	✗	✓	✓	Graph-powered Reinforcement Learning for Vehicular Task Offloading (GRLVTO), prioritizing high-resource-demand tasks, pbest match server
[5]	✗	Dynamic	SUMO, OMNet++	-	✓	✗	✓	✗	COADRL, Heuristic Greedy Offloading Scheme (HGOS), Task Outsourcing Only (TOSO), Task Outsourcing Then Relocation (TOSM), Task Native Processing (TNP)
[114]	✗	Dynamic	SUMO	✓	-	✗	✓	✓	MESON, Local, Greedy, Q-Learning, SA-DQN, MGA
[43]	✗	Dynamic	MATLAB	-	✓	✗	✓	✗	RSU-assisted Learning-Based Task Offloading Algorithm (RALPTO), UCB, volatile UCB (VUCB), genie-aided policy, random policy

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[16]	✗	Dynamic	Python, Directed Acyclic Graph Generator (DAGGEN)	✓	-	✓	✓	✗	Seq2Seq- based Meta Reinforcement Learning (SMRL-MTO), PPO, DQN, Greedy
[34]	✗	Dynamic	Python	✓	-	✗	✓	✗	Multi-Agent Proximal Policy Optimization-Based Task and Target Selection Algorithm (MAPPO-TTSA), DDQN, PPO, MEC Server Execution (ME), Local Execution (LE), Random Offloading (RO)
[46]	✗	Dynamic	Python	✓	-	✓	✓	✗	ODCO, Actor-Critic Algorithm (AC), DQN, Distributed Earliest-Finish Time Offloading (DEFO), random offloading

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[99]	✗	Dynamic	-	✓	-	✗	✓	✓	Multi-Layer D3QN and PDQN-Based Algorithm (MLDPBA), Multi-Layer D3QN-Based Algorithm (MLDBA), Single-Layer Deep-Based Algorithm (SLDBA), GBA, Random Based Algorithm (RBA), Local Based Algorithm (LBA)
[98]	✗	Dynamic	MATLAB	✓	✓	✗	✗	✓	MEGAN, MCSS, COMA-based
[52]	✗	Dynamic	-	-	✓	✗	✓	✓	Tabular SORL, Deep SORL
[93]	✗	Dynamic	-	✓	-	✗	✓	✓	R-DDQN, DQN, Mathematical Calculation
[116]	✓	Dynamic	Python	✓	-	✗	✓	✗	Meta-Policy based Task Offloading (MPTO), Greedy, Only Local, Only MEC, PPO

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[102]	✗	Dynamic	Python	✓	-	✗	✓	✗	DDPG-based edge caching and offloading, Random edge caching and offloading, Executing all tasks in the cloud, Least recently used (LRU) edge caching and offloading, Offloading without edge caching
[108]	✗	Dynamic	-	✓	-	✗	✓	✗	A3C, Random, DQN, Actor-Critic (A2C)
[50]	✗	Dynamic	-	✓	-	✗	✓	✗	DIOW, Delay-Greedy Optimization (DGO), No BS Peer Offloading (NoBP)
[7]	✗	Dynamic	-	✓	-	✗	✓	✗	AODAI, ACTO-n, DQN, State Action Reward State Action (SARSA), Double DQN, Dueling DQN, Shortest Latency (SL), Shortest Tolerance Time (ST), Reinforce-based approach (Rf-n)
[83]	✗	Dynamic	-	-	-	✗	✓	✗	K-means, Differential Evolution (DE), Nelder-Mead (NM)

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[12]	✗	Dynamic	-	✓	-	✗	✓	✗	FL-TD3, TD3, DDPG
[101]	✗	Dynamic	Python	✓	-	✓	✓	✗	Improved SAC, DQN, Q-learning
[8]	✗	Dynamic	-	✓	✓	✗	✓	✓	Sliding-Window UCB (SW-UCB), UCB, E-Greedy, Random, TOD, Off-Policy
[89]	✗	Dynamic	Python	✓	-	✗	✓	✗	Actor-Critic, DQN
[86]	✗	Dynamic	-	✓	-	✓	✓	✗	Dynamic Framing Offloading (DFO)-DDQN, DFO-DQN, DFO-Dueling DQN, Task Segmentation (Seg)-Greedy, Noseg-Greedy
[65]	✓	Dynamic	MATLAB	✓	-	✗	✓	✓	Model-Based DNN, Without DNN, Greedy, Always-Migrate, Random
[92]	✓	Static & Dynamic	Python	✓	-	✗	✓	✓	Bound-and-Bound based Optimal (BBO), Heuristic, Revenue_First, DRL, Random, Revenue-to-Cost (R2C)_First

(Table 2.2 continues on next page)

Paper	Theoretical Limit	Offloading Mode	MTG	Training		KPI			Algorithm
				Offline	Online	AET	Latency	DTN	
[109]	✗	Dynamic	SUMO	✓	-	✓	✓	✗	Enumeration, Deep Deterministic Policy Gradient-based Task Offloading Algorithm (DDPGTO), Game theory-based, DDPGTO without V2V, Local execution
Ours	✓	✓	SUMO	✓	-	✓	✓	✓	FCFS, SDF, Off-Sta-PSO, On-Sta-PSO, On-Dyn-PSO, On-Dyn-CDA, DQN, PPO

** MTG: Mobility Trace Generator, KPI: Key Performance Indicator, AET: Algorithm Execution Time, DTR: Dropped Tasks Ratio*

Chapter 3

A Reliable and Efficient 5G Vehicular MEC: Guaranteed Task Completion with Minimal Latency

This chapter explores the advancement of VEC as a tailored application of MEC for the automotive industry, addressing the rising demand for real-time processing in Connected and Autonomous Vehicles (CAVs). VEC brings computational resources closer to vehicles, reducing data processing delays crucial for safety-critical applications such as autonomous driving and intelligent traffic management. However, the challenge lies in managing the high and dynamic task load generated by vehicles' data streams. We focus on enhancing task offloading and scheduling techniques to optimize computation latency in VEC networks. Our approach involves implementing task scheduling techniques, including FCFS, SDF, and PSO. Additionally, we divide portions of tasks between the MEC servers and vehicles to reduce the number of dropped tasks and improve real-time adaptability. This chapter also compares fixed and shared bandwidth scenarios to manage transmission efficiency under varying loads. Our findings indicate that MEC+Local scenario significantly outperforms MEC-only scenario by ensuring the completion of all tasks, resulting in a zero task drop ratio. The MEC-only scenario demonstrates approximately 5.65% better average E2E latency compared to the MEC+Local scenario when handling 200 tasks. However, this improvement comes at the cost of dropping a significant number of tasks (109 out of 200). Additionally, allocating shared bandwidth helps to slightly decrease transmission waiting time compared to using fixed bandwidth.

3.1 Introduction

MEC has gained increasing importance in today's technology landscape, where the number of connected devices continues to grow. By bringing computational power closer to where data is generated, MEC minimizes the need to send information to distant cloud servers, reducing data processing delays [23]. Building on MEC's benefits, VEC tailors this concept specifically for the automotive and transportation industries. VEC brings processing capabilities closer to vehicles, enabling faster handling of data from sensors and communication systems in connected vehicles [110]. This edge-based processing is crucial for applications like autonomous driving, where real-time decision-making is vital for safety [49]. By providing essential computing resources at the network's edge, VEC ensures that connected vehicles operate more safely, efficiently, and reliably within today's transportation networks [114]. Vehicles continuously send data that require fast processing, which can overwhelm edge servers and cause communication and processing delays [6]. Inadequate task management may result in task drops by servers; however, optimizing the task queue can alleviate these delays by establishing the most efficient processing sequence.

MEC+Local approach also improves the efficiency of the VEC by dividing tasks between the MEC servers and the vehicles themselves [33]. This approach leverages computing resources at both the edge and in vehicles to reduce processing time and improve flexibility. This balanced approach reduces task drops and ensures that VEC systems handle the workload effectively. The goal is to offload as many tasks as possible to the VEC servers without overwhelming them, which could result in some tasks not being processed on time. By balancing local and edge processing, MEC+Local approach reduces communication and computation delays, increasing the number of tasks that are completed on time. This approach enhances system efficiency, supporting the advancement of sophisticated VEC applications designed to meet the needs of modern transportation [25] [115]. The main contributions of this work are as follows:

1. Our scenario includes strict deadlines based on RSU coverage and large tasks, offering a more accurate assessment of network performance under high-stress conditions.

2. Simultaneous tasks can share channels for uplink and downlink, thereby reducing transmission waiting times.

3. By tracking and analyzing dropped tasks within the RSU, the system dynamically adjusts parameters to reduce task loss and optimize resource use.

Section II presents the literature review, Section III the system model, Section IV the performance analysis, and Section V the conclusion.

3.2 Related Work

Recent studies have introduced a variety of strategies to improve task offloading in MEC for vehicular networks. Li et al. [39] investigate task offloading using DRL within MEC-supported heterogeneous vehicular networks, utilizing both V2I and V2V communications. They aim to maximize computation rates using serial and parallel offloading schemes. However, their study mainly focuses on computation optimization, with limited emphasis on communication resource management, potentially causing bottlenecks in high-load environments. Moreover, their model lacks scalable MEC+Local MEC+Local approach and effective dropped task management, which are crucial for high-demand, time-sensitive applications. Similarly, in another study, the authors propose a MEC model assisted by NOMA that emphasizes computation resource management in mobility-aware networks [41]. By considering vehicle speeds and task arrival rates, they work to optimize task offloading performance. Although they employ NOMA for channel sharing, their model's scalability is limited, assuming a fixed setup where each subchannel is shared by only two vehicles. Additionally, the model does not address task partitioning or advanced communication strategies that could adapt to varying traffic conditions.

Another study presents a DRL-based collaborative framework for vehicular networks, using DDPG techniques to improve task offloading and reduce service delays by distributing tasks in parallel [42]. Despite this model's adaptability to dynamic vehicular environments, it still lacks flexible communication resource allocation and options for local processing, which could enhance its adaptability and decrease delays in constrained network scenarios. Building on previous work, Moon and Lim [59] propose a framework that focuses on task partitioning and migration across MEC servers to improve load balancing. Their model effectively reduces task delays by migrating tasks from overloaded servers to those with lighter loads; however, it still lacks a flexible approach to communication resource management. Similar to [42], it does not consider the potential for local processing within vehicles, which could provide added flexibility when MEC resources are under strain.

Further research on MEC optimization explores joint computation offloading and resource allocation, incorporating partial and MEC-only offloading schemes to reduce latency [44]. While task partitioning proves beneficial for latency reduction, the model's dependence on Orthogonal Frequency-Division Multiple Access (OFDMA) restricts communication flexibility and does not include management of the number of dropped tasks. Expanding upon previous studies, a joint optimization framework is presented that addresses offloading, resource allocation, and data caching with the goal of minimizing processing costs [17]. Utilizing a Binary PSO algorithm, they target tasks that need both computation and data caching. Nonetheless, the model does not include dynamic channel sharing or number of dropped tasks considerations.

To address these existing limitations, we aim to develop a more realistic simulation that models scenarios with strict deadlines and large task sizes. In addition to optimizing computation, we enhance communication processes for both uplink and downlink, considering the impact of dropped tasks to enhance the overall performance. For the communication aspect, we manage bandwidth sharing by enabling tasks that arrive simultaneously within the RSU range to share bandwidth during uplink, as well as for tasks that are ready for parallel downlink. This approach aims to reduce transmission waiting time, thereby improving efficiency across the network.

3.3 System Model

This section provides an overview of the offloading schemes, including the applied methods and the mathematical formulation of the problem.

3.3.1 The Offloading Schemes

We explore scheduling algorithms including FCFS, SDF, and a heuristic approach, PSO. FCFS processes tasks strictly by arrival order, offering simplicity and predictability but ignoring task deadlines [60]. This can delay time-sensitive tasks and lead to inefficiencies, increased latency, and task drops, especially under heavy traffic. The SDF scheduling method prioritizes tasks based on deadlines, with shorter deadlines processed first. Deadlines depend on the distance between vehicles and the RSU, allowing tasks with less remaining time to be prioritized and reducing dropped tasks. However, SDF can delay tasks with longer deadlines, especially in high-traffic situations.

Our optimization approach includes both MEC-only and MEC+Local approaches. In the MEC-only method, the entire task is offloaded to a MEC server, whereas MEC+Local approach divides the task such that some components are processed remotely while others are handled locally on the vehicle. This approach is beneficial when a significant number of tasks are dropped in the MEC-only scenario. Based on Figure 5.1, the offloading scenario is simulated using SUMO. In this simulation, vehicles offload computationally intensive tasks to the RSU when they enter its communication range. Task deadlines are defined by the duration that vehicles remain within the RSU's coverage. Based on decisions made by the PSO algorithm, tasks are partitioned, with some portions offloaded to the RSU and the remainder processed locally.

PSO determines the optimal queue position for each task and assigns it to the most suitable MEC server. The algorithm focuses on minimizing overall latency and reducing

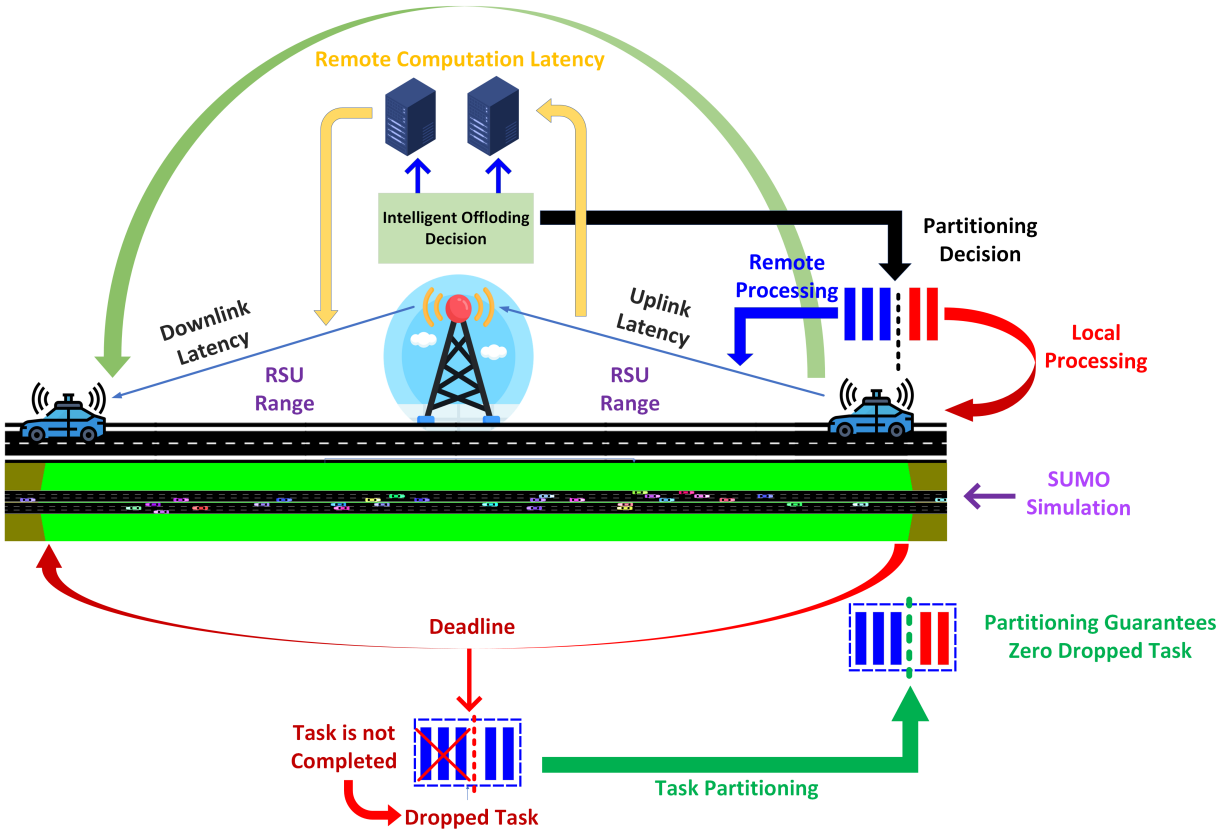


Figure 3.1: Problem statement for MEC+Local and MEC-only scenarios in 5G Vehicular MEC

the number of dropped tasks by evaluating factors such as task deadlines and server availability. Based on PSO decisions, portions of tasks that cannot be processed and returned to the vehicle before it leaves the RSU's range are either processed locally in MEC+Local approach or dropped in MEC-only scenario. This decision-making process takes place at the RSU. The communication latency includes both the time taken to send tasks from the vehicles to the RSU, and the time taken for the vehicles to receive results from the RSU. Moreover, we consider two scenarios for bandwidth allocation. In the first scenario, all tasks have a fixed maximum bandwidth, so only one task is offloaded at any given time. In the second scenario, when multiple vehicles arrive within the RSU range with tasks to be offloaded, those tasks are offloaded simultaneously by sharing the available bandwidth. These approaches are also considered in the downlink phase, where results are transmitted back to vehicles from the MEC servers.

Table 3.1: Notation Table

Param.	**Description**
L_i^{cm}	Communication latency for task i
T_i^{cm}	Communication time for task i
W_i^{u}	uplink waiting time for task i
W_i^{d}	downlink Waiting time for task i
S_i	Size of task i
r_i	Transmission rate for task i
B_i	Bandwidth of task i
p	Transmission power
g	Channel power gain
n_0	Noise power
t_i^{a}	Arrival time of task i on the RSU
$t_i^{\mathrm{a}'}$	Arrival time of task i to the vehicle
t_i^{r}	The time at which task i is ready to be offloaded to the RSU
$t_i^{\mathrm{r}'}$	The time at which task i is finished processing and ready to be sent back to its vehicle
t_i^{sp}	Start processing time for task i
$B_{\max}$	Maximum bandwidth
L_i^{p}	Remote computation latency for task i
T_i^{p}	Remote computation time for task i
T_i^{pl}	Local computation time for task i
W_i^{p}	Remote computation waiting time for task i
n	Total number of tasks
n'	Number of tasks that arrive at the same time at RSU range or finish processing on each MEC server simultaneously
m	Total number of MEC servers

(Table 3.1 continues on next page)

x_{ij}	Binary decision variable for assigning task i to MEC j
T_i^{range}	Remaining time in the RSU range for task i

3.3.2 Problem Formulation

All mathematical notations are provided in Table 3.1. As illustrated in Figure 3.1, the E2E latency is the sum of communication, remote computation, and local computation latencies. The total communication latency is formulated in (3.1). Since the transmission time, T_i^{cm}, is identical for both uplink and downlink, given that the task output sizes are assumed to match their input sizes, T_i^{cm} is multiplied by 2 to account for both directions. T_i^{cm} is formulated in (3.2), with transmission rate, r_i, defined in (3.3).

$$L_i^{\text{cm}} = 2 \times T_i^{\text{cm}} + W_i^{\text{u}} + W_i^{\text{d}} \tag{3.1}$$

$$T_i^{\text{cm}} = \frac{S_i}{r_i} \tag{3.2}$$

uplink waiting time, W_i^{u}, is formulated in (3.4), which represents the time each task spends waiting in a vehicle until bandwidth is available.

$$r_i = B_i \cdot \log_2 \left(1 + \frac{p \cdot g}{n_0}\right) \tag{3.3}$$

$$W_i^{\text{u}} = t_{i-1}^{\text{a}} - t_i^{\text{r}} \tag{3.4}$$

In scenarios where a vehicle enters the coverage area while a prior task is still being transmitted to the RSU, it must wait until the bandwidth is totally available. However, if a task is the first to be offloaded or is ready after the previous task has fully arrived, W_i^{u} is zero. Furthermore, we consider downlink waiting time, W_i^{d}, for each MEC server when transmitting results back to the vehicles. As formulated in (3.5), if a task completes processing before the prior task has been fully delivered to the vehicle, it must wait until the bandwidth is available.

$$W_i^{\text{d}} = t_{i-1}^{\text{a}'} - t_i^{\text{r}'} \tag{3.5}$$

Similar to W_i^{u}, if a task is the first one or becomes ready to be sent back to the vehicle after the previous task has fully arrived at its vehicle, W_i^{d} is set to zero. In the fixed bandwidth scenario, we also consider the potential waiting time in conditions where tasks complete simultaneously on both servers. In such cases, one of the MEC servers experiences an additional delay of W_i^{d}, determined by the task on the other MEC server. In the fixed bandwidth scenario, a maximum bandwidth is allocated to all tasks during both uplink and downlink, while the shared bandwidth approach allows the bandwidth to be distributed among the tasks. According to (3.6), in the shared bandwidth scenario, for both uplink and downlink, tasks that enter the RSU range simultaneously, as well as those completed at the same time on both MEC servers, will share the available bandwidth and be transmitted concurrently. Whereas, other tasks utilize the maximum bandwidth as shown in (3.6).

$$B_i = \begin{cases} B_{\max} \cdot \frac{S_i}{\sum_{i=1}^{n'} S_i} & n' > 1 \\ B_{\max} & n' = 1 \end{cases} \tag{3.6}$$

where n' represents the number of tasks that either arrive simultaneously within the RSU range or are completed simultaneously on the MEC servers. Since there are only two MEC servers, n' is equal to 2 for the downlink process.

This set of simultaneous tasks is represented as:

$$\mathcal{N} \;=\; \left\{ i = 1, \ldots, n' \;\middle|\; \mathcal{T}_i = \mathcal{T} \right\} \tag{3.7}$$

where $\mathcal{T}i$ indicates the offloading time at which task i is ready for transmission, whereas $\mathcal{T}$ corresponds to the moment when every task in $\mathcal{N}$ becomes ready for offloading.

It is notable that tasks in each MEC server are completed sequentially, meaning that, in both scenarios, only one task from each MEC server utilizes the bandwidth at any given time. The computation latency for remote processing is represented as:

$$L_i^{\mathrm{p}} = T_i^{\mathrm{p}} + W_i^{\mathrm{p}} \tag{3.8}$$

$$W_i^{\mathrm{p}} = t_i^{\mathrm{sp}} - t_i^{\mathrm{a}} \tag{3.9}$$

Remote computation time, T_i^{p}, and local computation time, T_i^{pl}, for each task are determined based on remote and local inference times from [91]. The formula for computation waiting time, W_i^{p} is provided in (3.9).

The start processing time of the first and second tasks aligns with their arrival time at the RSU. This is because they are assigned to MEC servers as soon as they arrive. The start processing time, t_i^{sp}, is calculated as:

$$t_i^{\text{sp}} = t_{i-1}^{\text{sp}} + T_{i-1}^{\text{p}} \tag{3.10}$$

Where for subsequent tasks after the first and second tasks, their start processing time, t_i^{sp} is the sum of their previous task's start processing time, t_{i-1}^{sp} and processing time, T_{i-1}^{p}. Our objective is to reduce both the total E2E latency and the number of dropped tasks, as illustrated in (3.11). Our main focus is to first reduce the number of dropped tasks and then minimize the E2E latency of the remaining ones.

$$\min \left(\lambda(\sum_{j=1}^{m} \sum_{i=1}^{n} L_i^{\text{e2e}} \times x_{ij}) + (1-\lambda)\text{D} \right) \tag{3.11}$$

Where λ is the weight for E2E latency, while $1-\lambda$ represents the weight for the number of dropped tasks. L_i^{e2e} and D are defined as:

$$L_i^{\text{e2e}} = p_i \cdot (L_i^{\text{cm}} + L_i^{\text{p}}) + (1-p_i) \cdot T_i^{\text{pl}} \tag{3.12}$$

$$D = \frac{1}{n} \sum_{i=1}^{n} \Big(1 - \sum_{j=1}^{m} x_{ij} \Big) \tag{3.13}$$

x_{ij} is indicated as:

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to MEC } j \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

Also, p_i represents the fraction of a task that is offloaded to a MEC server, while $(1-p_i)$ denotes the portions processed locally. In MEC-only offloading, p_i is set to 1, indicating that the entire task is offloaded with no local processing. Our PSO algorithm designates p_i along with task indices, specifying their processing order. For MEC+Local scenario, we have x_{ij}=1 and $0 \leq p_i \leq 1$. Additionally, the PSO determines the allocation of each task to a specific MEC server. According to constraint (3.15), each task is processed by only one MEC. Furthermore, constraint (3.16) ensures a task is only assigned to a server if the expected E2E latency is less or equal to remaining time in the range.

$$\sum_{j=1}^{m} x_{ij} \leq 1 \quad \forall i \in n \tag{3.15}$$

$$1 - \left(\frac{L_i^{\text{e2e}}}{T_i^{\text{range}}} \right) \leq x_{ij} \tag{3.16}$$

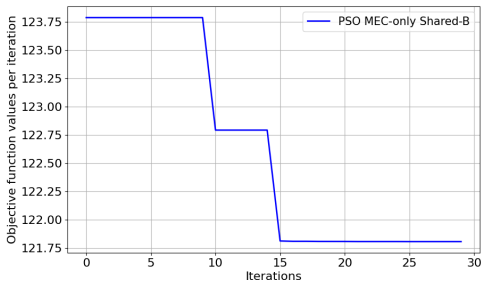
This helps in managing the Quality of Service (QoS) by ensuring that the E2E latency does not exceed the remaining time in range for each task.

3.4 Performance Analysis

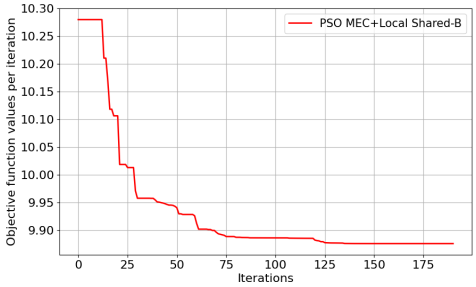
In this study, we simulate a highway scenario using the SUMO platform, where a RSU supports computational offloading for vehicles traveling along the highway. The RSU is equipped with two MEC servers, designed to handle incoming tasks from vehicles in its coverage area. We incorporate a time-sensitive requirement for each task, with deadlines determined by the Euclidean distance between the vehicle and the RSU at the time of task offloading. The simulation covers three different scenarios with varying traffic densities to evaluate MEC server performance under different workloads. Each vehicle generates a single image-processing task. Tasks are generated based on a Poisson distribution to simplify the model. Tasks can be generated both when a vehicle is within the RSU's range and any time prior to entering this range, which enables a more realistic model of real-world scenarios. The size of each task is determined by the resolution of the image being offloaded. We simulate different image resolutions to represent varying levels of computational load.

Table 3.2: Simulation and PSO Parameters Values

Parameters	**Value**
Number of vehicles	50, 100, 200
Number of tasks per vehicle	1
Max bandwidth	20 Mhz
Noise power	100 dbm
Transmission power	200 mW
Number of MEC servers	2
Number of CPU in each server	1
λ	0.3
Swarm Size	50
Maximum number of iterations	100
Personal & global Learning Coefficient	0.5



(a) MEC-only Scenario



(b) MEC+Local Scenario

Figure 3.2: Convergence Plots for PSO with Shared Bandwidth for 200 tasks

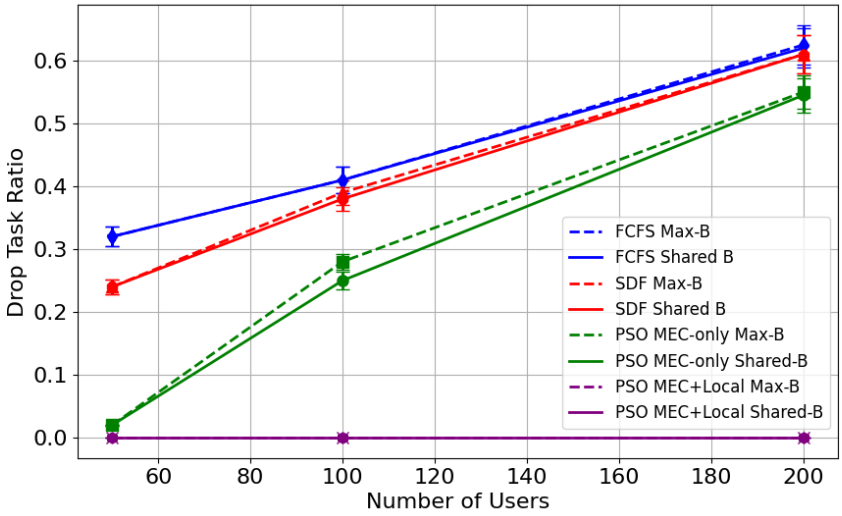


Figure 3.3: Average Drop task ratio for different number of vehicles

The parameters for simulation and PSO are listed in Table 3.2. The maximum bandwidth is 20 MHz, but after accounting for a 4.6% guard band, the effective usable bandwidth is 19.08 MHz. Figure 3.2a and Figure 3.2b present the convergence plots for PSO in the MEC-only and MEC+Local scenarios, illustrated for the maximum number of vehicles and shared bandwidth scenario. The number of iterations in each figure varies due to early stopping. For better comparison, we present the ratio of dropped tasks in Figure 3.3, defined as the number of dropped tasks divided by the total number of tasks. The figure demonstrates that as the number of tasks increases, preventing task drops becomes challenging due to short deadlines and high waiting times caused by the larger task load.

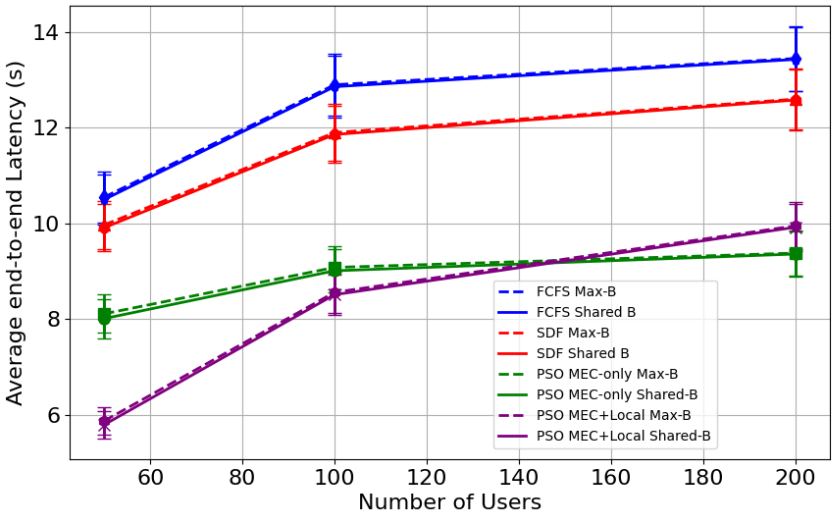


Figure 3.4: Average E2E latency for different number of vehicles

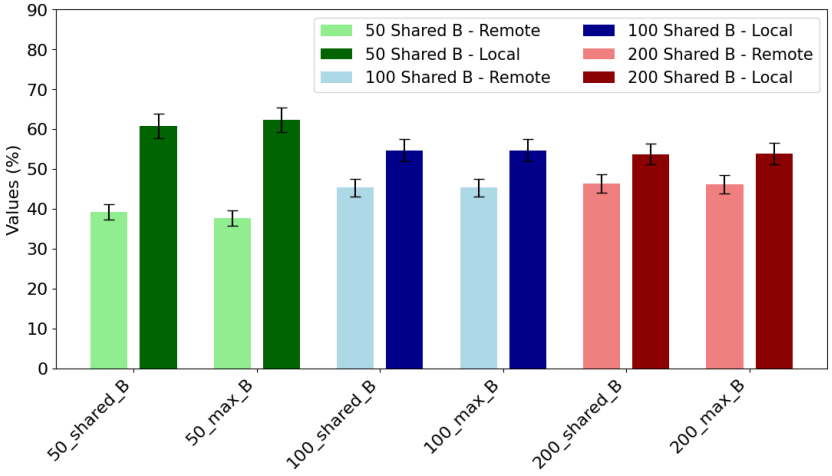


Figure 3.5: Average local and remote portions for different number of vehicles

To address this, we implement MEC+Local approach . As shown, in the MEC+Local approach, no tasks are dropped, indicating that PSO successfully identifies the optimal number of portions to offload, ensuring tasks meet their deadlines while maximizing the offloaded portions as much as possible. Figure 3.4 displays the average of E2E latency for 10 runs. As shown, with an increasing number of tasks, MEC+Local scenario does not reduce the latency compared to MEC-only offloading. This result arises from the significant

difference in numbers of dropped tasks between MEC-only and MEC+Local approaches, with a difference of 109 tasks out of 200. As dropped tasks' latencies are not included in the E2E latency, latency is lowered in the MEC-only scenario. In FCFS and SDF, where the order of tasks is handled simply, both the number of dropped tasks and E2E latency are less beneficial compared to PSO. In both approaches, PSO outperforms FCFS and SDF due to its optimized task ordering. As illustrated in Figure 3.5, most portions are processed locally due to the significant waiting times experienced on the MECs. Without incorporating computational waiting times into the objective function, we could have more remote portions than local ones, though this would lead to increased E2E latency, highlighting why it is important to consider waiting times in our objective function.

3.5 Conclusion

In conclusion, this study shows that task optimization and partitioning improve VEC performance, with shared bandwidth among vehicles offering slight benefits over fixed maximum bandwidth. Findings indicate that, for the MEC-only scenario with 200 tasks using the shared bandwidth approach, 109 tasks are dropped, whereas 0 tasks are dropped in the partitioning approach. This highlights the efficiency of the partitioning approach in managing task distribution compared to the MEC-only scenario. This significant number of dropped tasks in the MEC-only scenario results in a lower average E2E latency (5.65% reduction) since only successfully completed tasks are considered. However, despite this, the E2E latency in the partitioning approach is still lower compared to the FCFS and SDF methods. In the future work, we plan to incorporate energy optimization and implement more advanced algorithms to enhance performance and improve system efficiency. Energy optimization is essential to reduce power consumption, particularly in resource-constrained environments, while advanced algorithms can further optimize the overall offloading.

Acknowledgment

This work was supported in part by funding from the Innovation for Defence Excellence and Security (IDEaS) program from the Department of National Defence (DND), in part by Natural Sciences and Engineering Research Council of Canada (NSERC) CREATE TRAVERSAL Program, and in part by the NSERC DISCOVERY Program.

Chapter 4

On-Dyn-CDA: A Real-Time Cost-Driven Task Offloading Algorithm for Vehicular Networks with Reduced Latency and Task Loss

This chapter addresses the critical challenge of real-time task processing in vehicular networks, where timely and efficient task execution is crucial. Our primary objective is to maximize the number of completed tasks while minimizing overall latency, with a particular focus on reducing number of dropped tasks. To this end, we investigate both static and dynamic versions of an optimization algorithm. The static version assumes full task availability, while the dynamic version manages tasks as they arrive. We also distinguish between online and offline cases: the online version incorporates execution time into the offloading decision process, whereas the offline version excludes it, serving as a theoretical benchmark for optimal performance. We evaluate our proposed On-Dyn-CDA against these baselines. Notably, the static PSO baseline assumes all tasks are transferred to the RSU and processed by the MEC, and its offline version disregards execution time, making it infeasible for real-time applications despite its optimal performance in theory. Our novel On-Dyn-CDA completes execution in just 0.05 seconds under the most complex scenario, compared to 1330.05 seconds required by Dynamic PSO. It also outperforms Dynamic PSO by 3.42% in task loss and achieves a 29.22% reduction in average latency in complex scenarios. Furthermore, it requires neither a dataset nor a training phase, and its low computational complexity ensures efficiency and scalability in dynamic environments.

4.1 Introduction

MEC has emerged as an essential component in the era of IoT, where the number of connected devices, such as smartphones, sensors, and smart home appliances, is rapidly increasing [67], [23]. This proximity reduces dependency on distant centralized cloud servers, significantly lowering latency [73].

Building on the benefits of MEC, VEC extends this paradigm to meet the specific demands of CAVs and intelligent transportation systems [6, 38]. As vital elements of the IoT ecosystem, CAVs generate vast amounts of data from sensors, which must be processed in real time [7], [64]. ML models play a crucial role in processing this sensor data, enabling CAVs to accurately make decisions that ensure safe and efficient driving [72], [104]. However, due to the immense computational demands, onboard processing alone is often insufficient to meet real-time requirements [95], [107], [55]. VEC provides a solution by enabling vehicles to offload computationally intensive tasks to nearby edge servers [27], [30]. CAVs also rely heavily on communication technologies, specifically V2V and V2X communication, to exchange critical information [15]. V2V communication enables vehicles to share data, facilitating cooperative driving and collision avoidance [81]. Meanwhile, V2X extends this communication to include infrastructure, such as traffic lights and RSUs, as well as other connected devices [4].

Despite VEC's advantages, real-time vehicular environments face challenges as multiple vehicles offload tasks, causing server queues, delays, and dropped tasks. Optimizing task processing order is crucial to maximize successful task completion and minimize the latency [113], [33]. Conventional scheduling methods may not be adequate for these dynamic and high-load environments, making optimization algorithms, such as PSO, a promising approach [62]. However, traditional optimization algorithms like PSO often require considerable time to converge, which is impractical for real-time applications [94]. Consequently, developing a fast and reliable scheduling algorithm is essential to ensure that task offloading remains efficient, even in highly dynamic and demanding vehicular networks [79]. This need for rapid and real-time approaches underscores the importance of innovative techniques to maintain low latency and high reliability in VEC environments.

The main contributions of this work are as follows:

1. On-Dyn-CDA is proposed for efficient real-time task scheduling in vehicular networks. It adapts to continuous task arrivals and resource changes, making it suitable for practical use while handling many tasks with low computational cost.

2. We develop a dynamic optimization approach to schedule tasks within decision win-

dows, including tasks arriving before MEC availability, while accounting for the algorithm's execution time in each decision window.

3. Our study defines the theoretical upper bound, Off-Sta-PSO, which assumes that all tasks are available for optimization in advance, enabling ideal scheduling without accounting for the execution cost of the optimization itself, which would otherwise directly impact task processing. We also compare our model to a real-time worst-case scenario, On-Sta-PSO, without dynamic behavior, showing our proposed model performs close to the theoretical optimum and far better than the worst-case baseline.

Based on our contributions, our findings indicate that the novel dynamic algorithm, On-Dyn-CDA, efficiently manages real-time task scheduling in vehicular networks by dynamically adapting to continuous task arrivals and fluctuating resource availability with reduced computational cost. In our most complex environment of 200 tasks, it achieves approximately 95.58% of the theoretical limit for dropped tasks and outperforms dynamic PSO by about 3.42%. Furthermore, we show that On-Dyn-CDA reaches around 93.32% of the theoretical limit in terms of average latency and demonstrates a 29.22% improvement over dynamic PSO, with a significantly shorter execution time, validating its superior performance in real-world applications. Section II presents the literature review, while Section III outlines the methodologies under study. The performance evaluation is covered in Section IV, and Section V concludes the study.

4.2 Related Work

Several studies have examined task offloading and resource allocation in MEC-enabled IoV [14], [22], highlighting the challenges of meeting low-latency requirements when relying solely on traditional cloud computing. The proposed solutions employ multi-objective optimization methods, often focusing on latency and load balancing. Ant colony optimization can be one approach to address these challenges by exploring the search space for offloading decisions while balancing the computational load across available resources [84]. Additionally, multi-criteria decision-making and Simple Additive Weighing can be incorporated for comprehensive evaluation [84]. Dependent tasks are particularly critical in MEC-enabled 5G vehicular networks, as they often consist of multiple interlinked subtasks that must be processed in a specific sequence to maintain system efficiency. In resource-limited environments, these tasks can be modeled as DAGs, and an effective scheduling strategy is needed to meet stringent latency requirements [97]. This issue can be addressed by employing a PBTSA, utilizing RBFS to prioritize subtasks and leveraging a greedy offloading strategy for efficient task allocation, ultimately reducing overall delays [97].

Table 4.1: A comparison of our work with existing literature

Paper	# Tasks	# MECs	Dyn	Training		MTG	KPI			Algorithm
				Off	On		AET	Latency	DTR	
[84]	40-120	10	✗	-	-	-	✗	✓	✗	Ant Colony Optimization (ACO)
[97]	-	10	✗	-	-	Python	✗	✓	✗	PBTSA, PBLA, TBTOA
[58]	-	6	✗	-	-	-	✗	✓	✓	Mobility, Contact, and Computational Load-Aware (MCLA)
[41]	-	1	✗	-	-	MATLAB	✗	✓	✓	Theoretical analysis
[20]	50	4	✗	-	-	MATLAB	✗	✓	✓	Theoretical analysis
[29]	-	-	✓	✗	✓	SUMO + MATLAB	✗	✓	✗	long short-term memory (LSTM), PPO
[50]	-	4	✓	✗	✓	Python	✗	✓	✗	MADDPG
[90]	-	-	✓	✗	✓	Python	✗	✓	✗	DRL
[16]	-	-	✓	✓	✗	DAGGEN + Python	✗	✓	✗	SMRL-MTO
[106]	-	4, 32	✓	✓	✗	real historical dataset	✗	✓	✗	RL
Ours	50-100-200	2	✓	-	-	SUMO + python	✓	✓	✓	On-Dyn-CDA, Off-Sta-PSO, On-Sta-PSO, On-Dyn-PSO

** Dyn: Dynamic, Off: Offline, On: Online, MTG: Mobility Trace Generator, KPI: Key Performance Indicator, AET: Algorithm Execution Time, DTR: Dropped Tasks Ratio*

To improve task offloading in MEC-enabled vehicular networks, dynamic communication switching can adapt to network conditions. By leveraging V2V and V2I modes,

vehicles switch between sub-6 GHz 5G NR and mmWave to reduce delays and enhance execution rates. This approach optimizes offloading based on mobility, contact duration, and computational load, improving task turnover and network stability. Practical implementations report up to 1800 completed tasks, though the total number is not explicitly stated [58]. Beyond communication switching, efficient resource allocation is key to improving task offloading in MEC-enabled vehicular networks. Integrating NOMA enhances spectral efficiency by enabling simultaneous transmissions, reducing congestion, and improving execution rates. Analytical models show that vehicle arrival rates significantly impact task loss, while speed variations have minimal effect, emphasizing the need to prioritize traffic density in resource allocation [41].

Building on the advantages of NOMA in vehicular task offloading, further optimizations enhance spectrum utilization and computational efficiency by enabling multiple Vehicular User Equipments (VUEs) to share wireless resources, improving system capacity for intensive applications. However, task offloading and resource allocation remain complex. A decomposition strategy addresses this by breaking the problem into two subproblems, solved with heuristic algorithms to ensure efficient task distribution, low latency, and balanced resource usage [20]. Apart from spectrum and computational efficiency, task offloading must also account for heterogeneous coverage, where 5G Base Station (BS)s and RSUs coexist. In such scenarios, vehicles dynamically offload tasks based on real-time load and proximity to edge units. Since direct BS load measurement is challenging, LSTM-based predictive models estimate load in advance for informed decisions. Offloading is formulated as a constrained optimization problem to minimize delays and balance workloads, with RL methods like PPO refining decision-making in dynamic network conditions [29].

Further enhancing task offloading efficiency, distributed intelligence can be leveraged to balance workloads dynamically across edge resources. MADDPG algorithms offer a promising approach by optimizing system performance while minimizing delays and adhering to BS energy constraints [50]. The effectiveness of task offloading depends on factors such as how frequently tasks are generated, the number of active users, and the length of the simulation. However, [50] does not explicitly mention the total number of tasks or the simulation duration, which can limit the reproducibility of results.

Building on RL approaches, Actor-Critic-based DRL can be used to optimize task offloading by considering factors such as vehicle-RSU distance, channel conditions, and resource availability. This framework enables dynamic task distribution across mobile vehicles and RSUs, effectively balancing latency, energy consumption, and resource allocation [90]. Further enhancing task offloading efficiency, SMRL-MTO can be employed to capture dependencies between subtasks and optimize offloading decisions. By integrating bidirectional gated recurrent units with an attention mechanism, this approach dynami-

cally prioritizes key input sequences. The model considers DAGs with around 12 subtasks executed across multiple servers, though exact numbers are not specified [16]. Additionally, a model-agnostic meta-learning framework enables rapid adaptation to new multi-task offloading (MTO) scenarios with minimal retraining [16]. To mitigate the high exploration costs of online RL, offline RL can be employed for asynchronous task offloading in MEC environments [106]. By utilizing real historical data, this approach trains a supervised environmental model on a static dataset, simulating edge computing dynamics to optimize policies without real-world interactions. In [106], scalability is specifically evaluated by increasing the number of edge servers from 4 to 32, though details about the simulation tool are not provided.

While existing studies address key aspects of vehicular task offloading, they often overlook critical factors such as dynamic task processing, the number of dropped tasks, and reporting of algorithm execution time. Some focus on task dropping or dynamic scheduling individually but fail to consider them together. This gap is significant, as low latency becomes less meaningful if many tasks are dropped before processing. The priority should be ensuring tasks are completed and returned before deadlines, followed by minimizing latency for successfully processed tasks. Additionally, algorithm execution time is crucial, as fast decision-making is essential for meeting deadlines. Even optimal algorithms are ineffective if they cannot schedule tasks quickly.

To tackle these challenges, we begin by analyzing the theoretical limit scenario using the PSO algorithm. Next, we implement a dynamic PSO approach that assigns tasks to MECs in real-time based on the optimal order, enabling continuous adaptation to fluctuating task arrivals and resource availability. However, this method is not efficient for real-time applications due to the high execution time required by optimization algorithms. To overcome this limitation, we introduce On-Dyn-CDA, which reduces the number of dropped tasks and latency while maintaining a short algorithm execution time. PSO is also implemented in a static online simulation to assess its computational cost across all the tasks. Moreover, On-Dyn-CDA outperforms ML algorithms like RL by being more efficient, as it eliminates the need for training or datasets, operates faster, and reduces overall complexity. Additionally, we compare our method against simple greedy algorithms that offer fast execution times, demonstrating the superior performance of On-Dyn-CDA in dynamic settings. Table 4.1 highlights the differences between our approach and the related studies.

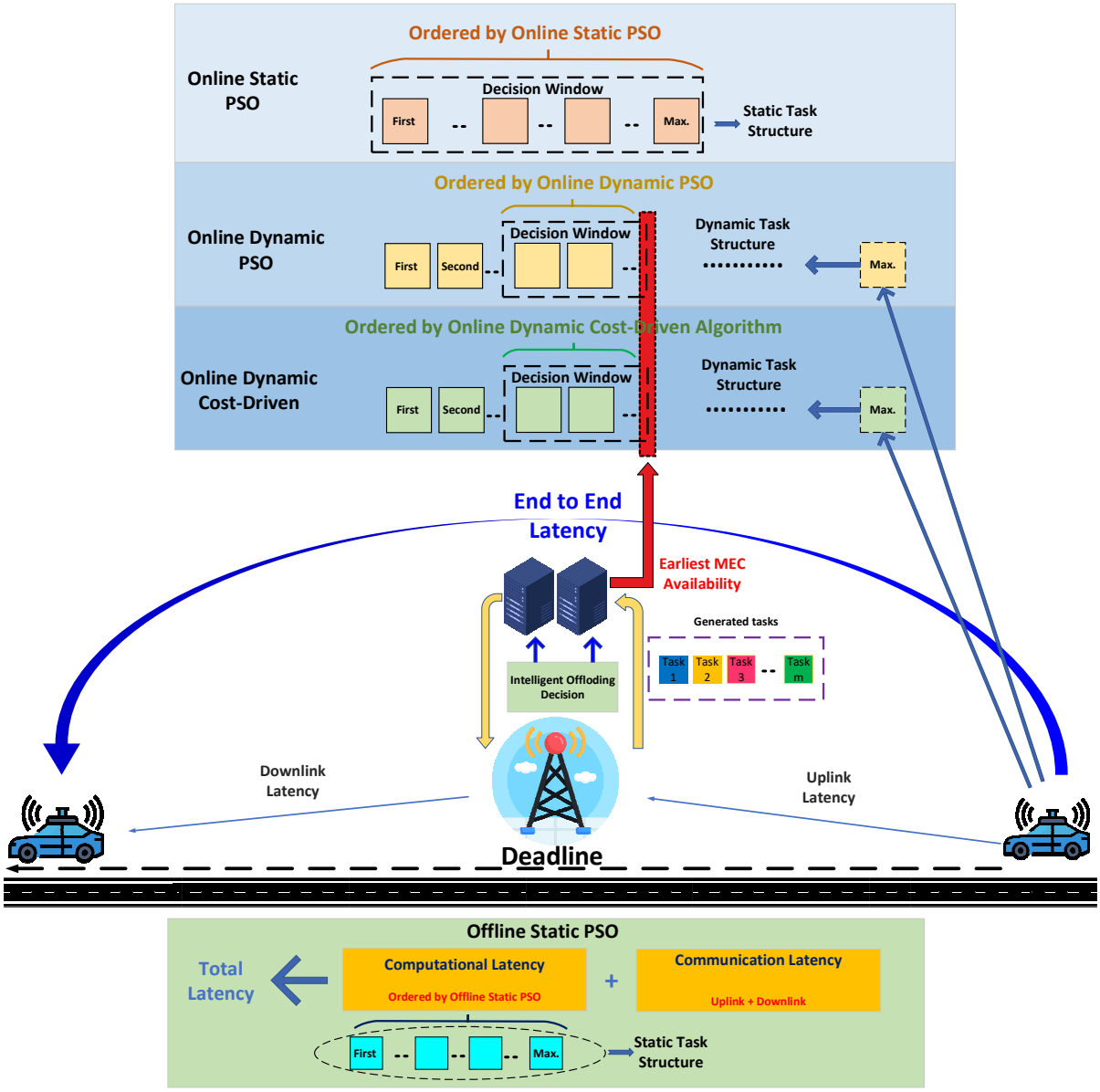


Figure 4.1: General Concept of VEC for Static and Dynamic Task Offloading Process

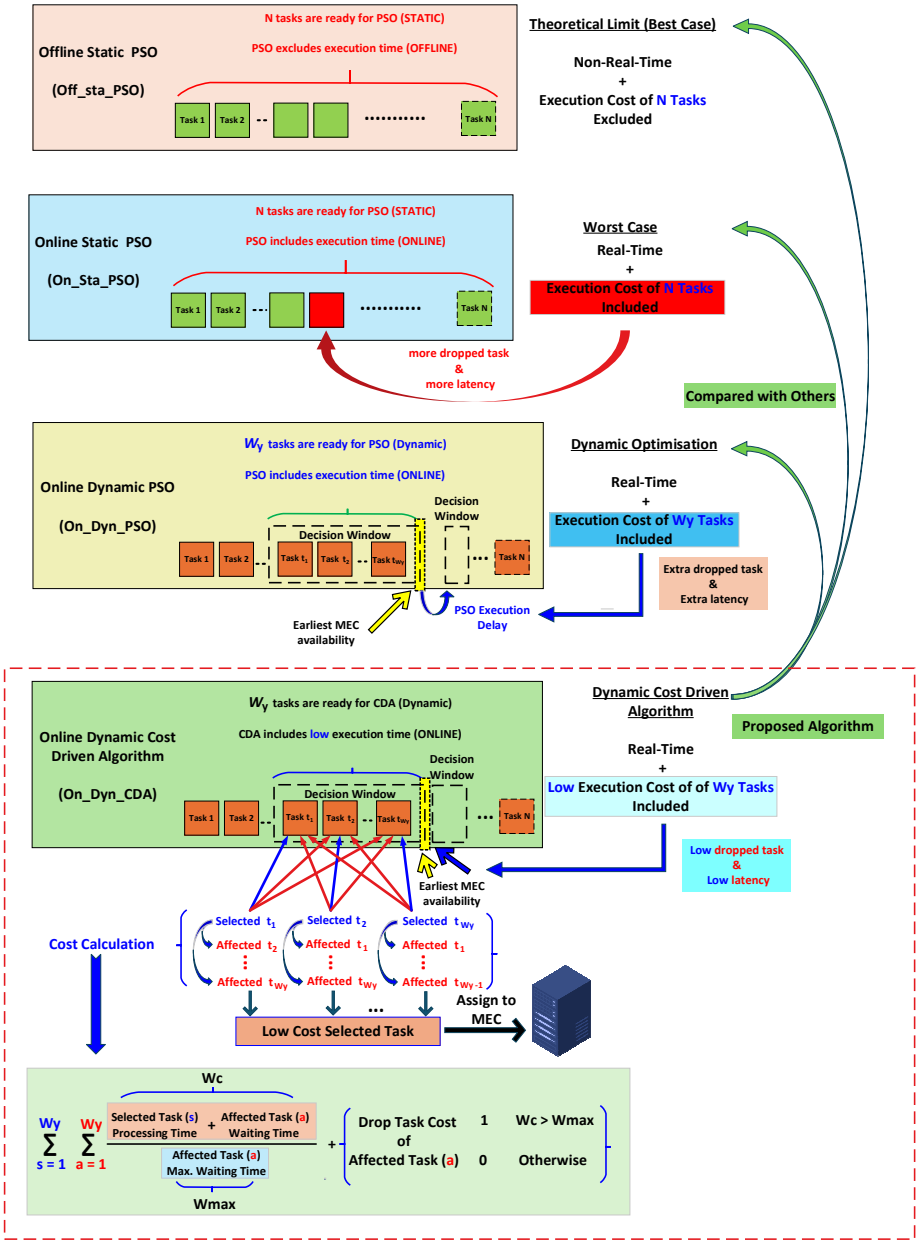


Figure 4.2: Comparative Analysis of the Proposed Algorithm and Existing Methods

4.3 Methodologies under Study

4.3.1 Baseline Methods and the Proposed Algorithm

We employ **FCFS** and **SDF** [60], both of which are greedy algorithms, along with **PSO**, a metaheuristic optimization technique that provides a more adaptive and efficient strategy for scheduling tasks in complex environments. In our approach, PSO determines the order in which tasks are processed and assigns each task to a specific MEC server. This assignment is guided by the goal of reducing the number of dropped tasks and minimizing the overall latency for tasks that are successfully processed. The algorithm evaluates key factors, such as task deadlines, server availability, and computational requirements, to ensure the efficient distribution of tasks across available resources.

Based on Figure 4.1 and Figure 4.2, three types of PSO algorithms are employed for comparison with our state-of-the-art approach: Off-Sta-PSO, On-Sta-PSO, and On-Dyn-PSO. Each algorithm addresses task scheduling under different assumptions and constraints, moving from idealized scenarios to more realistic ones. Off-Sta-PSO is designed to showcase the theoretical upper limit and the best possible solution. This algorithm assumes that we have prior knowledge of all tasks, including their arrival times and processing requirements, before they even arrive at the RSU. This is achieved by utilizing a pre-generated dataset from SUMO, representing a static task processing strategy. In this idealized setup, PSO is executed once on all tasks in the dataset to determine their order. Since there are two MECs, the start processing times for the first and second tasks are assumed to be equal to their arrival times, without considering any waiting time. As, in real-world scenarios, tasks arrive in real-time and their details are unknown before reaching the RSU, this assumption does not fully align with practical systems. However, it serves as a useful reference for establishing the optimal benchmark for task scheduling and performance.

On-Sta-PSO introduces a more realistic approach by calculating all waiting times in real-time. This still simulates a scenario in which the PSO algorithm is applied to all tasks at once. However, it assumes that all tasks must arrive at the RSU before the decision-making process begins, leading to significant waiting times for tasks that have already arrived. As a result, processing of the first and second tasks begins only after all tasks have been received, which reflects a worst-case scenario where processing is delayed to accommodate PSO execution after gathering complete task information. In both static algorithms, tasks one and two begin processing immediately upon arrival since two MECs are available. Subsequent tasks are then ordered using the PSO algorithm and start processing only after the preceding task is completed.

On-Dyn-PSO takes a dynamic, real-time approach, where PSO is executed multiple

times as tasks are generated dynamically by SUMO. In this scenario, tasks one and two are immediately assigned to MEC servers upon arrival. Subsequent tasks arriving after task two are collected in a queue called the decision window. This queue includes the set of tasks that arrive between the start processing time of task two and the earliest MEC availability time. We should note that, in static scenarios, the decision window includes all tasks in the simulation, as we optimize the scheduling of all tasks together at once. The earliest MEC availability time is defined as the moment when at least one MEC becomes free to process tasks. Until just before this availability time, the PSO algorithm is applied to identify the optimal task for assignment to the first available MEC. After assigning the task, the availability time of each MEC is updated, and the earliest MEC availability time is determined by taking the minimum of the MECs' availability times. A new decision window is created, ranging from the maximum start processing time of tasks across both MECs (corresponding to the start processing time of the most recently assigned task) to the new earliest MEC availability time. This decision window now includes all unassigned tasks that have already arrived or will arrive before the new earliest availability time. The PSO algorithm is then reapplied to identify the optimal task within this updated window, and the process continues iteratively until the simulation ends, ensuring that only one task is assigned to a MEC at a time. This approach enables dynamic, real-time task scheduling, closely reflecting how tasks would be managed in real-world scenarios. However, PSO requires considerable execution time, making it impractical for real-world scenarios. To address this limitation, we propose a faster, real-time solution called the On-Dyn-CDA. This proposed algorithm is designed to handle tasks efficiently in real-time by leveraging a cost function to determine which task should be assigned to the available MEC first. Similar to On-Dyn-PSO, tasks one and two are automatically assigned to MEC1 and MEC2 upon arrival. Additionally, as in On-Dyn-PSO, tasks arriving after the start processing time of task two and the earliest availability time are gathered in a decision window.

4.3.2 The mathematical framework for task offloading

E2E latency, a key component of the objective function, depends on both computation and communication latencies. Each is explained separately to highlight its individual impact.

computation latency

The start processing time of each task is calculated by adding the start processing time of the preceding assigned task to its processing time. The start processing time is essential for determining a task's waiting time, which contributes to its E2E latency. When task i

is offloaded to a MEC server j, it undergoes a delay composed of two main components: the time it waits before processing begins, and the actual processing duration. Based on (4.1), t_i^{w} represents the waiting time, indicating how long task i waits in the RSU before being processed.

$$t_i^{\mathrm{w}} = t_i^{\mathrm{sp}} - t_i^{\mathrm{ar}} \tag{4.1}$$

t_i^{sp} represents the start processing time of task i, and t_i^{ar} denotes the arrival time at the RSU for task i. Table 5.2 provides an explanation of all the necessary notations for the mathematical formulas. The computation latency, L_i^{p}, as expressed in (4.2), is defined as the total of the processing time and the waiting time.

$$L_i^{\mathrm{p}} = t_i^{\mathrm{p}} + t_i^{\mathrm{w}} \tag{4.2}$$

t_i^p denotes the processing time of task i with actual processing times sourced from [**?**].

Table 4.2: Notation Table

Param.	**Description**	**Eq.**
N	Total number of tasks	(4.9) (4.10) (4.13)
M	Total number of MEC servers	(4.9) (4.10) (4.11) (4.13)
N'	Number of tasks ready for transmission to/from RSU simultaneously	(4.4)
D	Total number of dropped tasks	(4.9) (4.10)
S_i	Size of task i	(4.4) (4.6)
x_{ij}	Binary variable indicating task i is assigned to MEC j	(4.8) (4.9) (4.10) (4.11) (4.13)
t_i^{range}	Remaining time in the RSU range for task i	(4.8)
t_i^{d}	The deadline for processing task i and returning it to the vehicle	(4.20)
t_i^{ar}	Arrival time of task i to the RSU	(4.1) (4.20)
t_i^{sp}	Start processing time for task i	(4.1) (4.11)

(Table 4.2 continues on next page)

Param.	Description	Eq.
L_i^{e2e}	End-to-end latency for task i	(4.7) (4.8) (4.10)
L_i^{p}	Computation latency for task i	(4.2) (4.7)
T_i^{p}	Processing time of task i	(4.2) (4.11) (4.20)
t_i^{w}	The time task i waits in the RSU before starting processing	(4.1) (4.2)
$t_i^{\text{w\_max}}$	The maximum duration that task i can wait before processing while meeting its deadline	(4.20) (4.22) (4.23)
L_i^{tr}	Transmission latency for task i	(4.7)
t_i^{tr}	Transmission time for task i	(4.6)
r_i	Transmission rate for vehicle of task i	(4.5) (4.6)
B_i	Bandwidth of task i	(4.4) (4.5)
$B_{\max}$	Maximum bandwidth	(4.4)
p	Transmission power	(4.5)
g	Channel gain	(4.5)
n_0	Noise power	(4.5)
t_j^{av}	The time when MEC j becomes available	(4.11) (4.12)
t_e^{av}	Earliest MEC availability time	(4.12)
t_s^{p}	The processing time of the selected task	(4.22) (4.23)
d_{sa}	Drop cost condition for an affected task based on the selected task	(4.22) (4.23)
S	The set of tasks available within the decision window	(4.15) (4.17)
Q	The set of tasks available within the decision window that meet theis deadlines	(4.17) (4.18) (4.19)
W_y	The number of tasks within the decision window that meet their deadlines	(4.17)

Transmission latency

During transmission, if multiple tasks within the RSU's range are offloaded simultaneously, the available bandwidth is shared among them. In such cases, the bandwidth allocated to each task is proportional to its size [70]. However, if a task is transmitted individually,

it can utilize the entire available bandwidth. Based on our study in [70], sharing the bandwidth can slightly reduce transmission latency compared to using a fixed bandwidth allocation. The set of concurrent tasks is defined as:

$$\mathcal{N} \;=\; \left\{\, i = 1, \ldots, N' \;\middle|\; \mathcal{T}_i = \mathcal{T} \right\} \tag{4.3}$$

$\mathcal{T}_i$ denotes the offloading time when task i becomes ready for transmission, while $\mathcal{T}$ represents the offloading time at which all tasks in $\mathcal{N}$ are ready to be offloaded. Additionally, N' represents the total number of tasks in set $\mathcal{N}$. The bandwidth assigned to task i is calculated using (4.4). This bandwidth is further used to calculate the transmission rate, r_i, which is needed to determine the transmission time.

$$B_i = \begin{cases} B_{\max} \cdot \frac{S_i}{\sum_{i=1}^{N'} S_i} & N' > 1 \\ B_{\max} & N' = 1 \end{cases} \tag{4.4}$$

where S_i represents the size of task i. The transmission rate is defined in (4.5):

$$r_i = B_i \cdot \log_2 \left(1 + \frac{p \cdot g}{n_0} \right) \tag{4.5}$$

where p indicating the transmission power, g referring to the channel gain, and n_0 denoting the noise power density.

Furthermore, the expression for the transmission time, t_i^{tr}, is provided in (4.6) and constitutes the transmission latency.

$$t_i^{\mathrm{tr}} = \frac{S_i}{r_i} \tag{4.6}$$

The downlink transmission follows the same process as the uplink for bandwidth allocation and calculation, and in our experimental results, their performance is very similar. Since the impact of transmission latency on total latency is relatively small, and for simplicity, we assume that the result size is equal to the input size.

As defined in (4.7), the end-to-end latency, L_i^{e2e}, includes both computation, L_i^{p}, and transmission, L_i^{tr}, latencies.

$$L_i^{\mathrm{e2e}} = L_i^{\mathrm{p}} + L_i^{\mathrm{tr}} \tag{4.7}$$

The transmission latency, L_i^{tr}, includes both the uplink transmission time and the downlink transmission time.

A binary variable, x_{ij}, is used to represent the assignment of task i to MEC server j as shown in (4.8). Task i is assigned to MEC j if its E2E latency does not exceed the duration the vehicle remains within range, t_i^{range} .

$$x_{ij} = \begin{cases} 1 & L_i^{e2e} \leq t_i^{range} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

Number of Dropped Tasks

The calculation for the number of dropped tasks, D, is presented in (4.9). This indicates the number of tasks that were not assigned to any MEC because they could not meet their deadlines, ranging from 0 to N:

$$D = \frac{1}{N} \sum_{i=1}^{N} \Big(1 - \sum_{j=1}^{M} x_{ij}\Big) \tag{4.9}$$

where M represents the total number of MECs.

The Optimization Problem

Our main objective is to minimize the number of dropped tasks, followed by reducing the E2E latency for non-dropped tasks. The objective function is defined in (4.10).

$$\min \left(\lambda (\sum_{j=1}^{M} \sum_{i=1}^{N} L_i^{\text{e2e}} \times x_{ij}) + (1 - \lambda) D \right) \tag{4.10}$$

λ is set to 0.4, indicating greater emphasis on D. This choice reflects the critical importance of task completion in our scenario, where dropped tasks represent failures of service. While a lower value of λ would place even more weight on avoiding task drops, we selected this value based on preliminary empirical observations balancing both objectives.

In the dynamic scenario, the first and second tasks are assigned to MEC1 and MEC2 immediately upon arrival at the RSU. After this initial assignment, we calculate the time each MEC becomes available, as demonstrated in (4.11).

$$t_j^{\text{av}} = (t_i^{\text{sp}} + t_i^{\text{p}}) x_{ij} \quad \forall j \in M \tag{4.11}$$

This indicates that each MEC becomes available, t_j^{av}, only after completing the task assigned to it.

To determine which MEC to assign the new task to, we identify the earliest MEC availability time, t_e^{av}, using:

$$t_e^{\text{av}} = \underset{j}{\text{argmin}} \left(t_j^{\text{av}} \right) \tag{4.12}$$

As outlined in constraint (4.13), every task is exclusively processed by a single MEC server.

$$\sum_{j=1}^{M} x_{ij} \leq 1 \quad \forall i \in N \tag{4.13}$$

Figure 4.3: Illustration of a basic scenario showcasing the functionality of the On-Dyn-CDA.

The Cost formula

As shown in (4.14), T is the set of all tasks in the simulation, containing a total of N tasks.

$$T = \{t_1, t_2, t_3, \ldots, t_N\} \tag{4.14}$$

After assigning a task to a MEC server, a new set of tasks, S, is formed as defined in (4.15). This set includes tasks that arrive before the earliest availability time, t_e^{av}, and have not yet been assigned to any MEC server.

$$S = \{t_{i_1}, t_{i_2}, t_{i_3}, \ldots, t_{i_W}\} \tag{4.15}$$

This implies that the arrival time of tasks, t_{il}^{ar}, in S is less than or equal to the earliest availability time, t_e^{av}, based on (4.16).

$$t_{il}^{ar} \leq t_e^{av} \tag{4.16}$$

The set Q_y is a subset of tasks from S that satisfy the condition of being able to meet their deadlines, as determined in (4.17). This means that only tasks that arrive before t_e^{av} and can meet their deadlines are included in the Q_y set, which represents our decision window. Tasks that cannot meet their deadlines are excluded from this window, as they are not eligible for assignment:

$$Q_y = \{t_{y_l} \in S \mid 1 \leq l \leq W_y,\ t_{y_l}^{ar} \leq t_e^{\text{av}},\ t_{y_l}^{w} \leq t_{y_l}^{range} - t_{y_l}^{p} - t_{y_l}^{\text{cm}}\} \tag{4.17}$$

where t_{y_l} denotes the task at position l in window y. As demonstrated in Figure 4.3, if there are W_y tasks in Q_y (with $W_y = 3$ illustrated in Figure 4.3), there are W_y possible tasks that can be selected to be assigned first to a MEC server once it becomes available. In other words, each task within the decision window that belongs to set Q_y can be selected to be sent to the MEC server with the earliest availability.

Our goal is to choose the task that minimizes overall latency and results in fewer tasks being dropped. As specified in (4.18), each task within the decision window's Q_y set is evaluated as the selected task, t_s, while the remaining tasks are treated as affected tasks and are included in the set Q_{ya}, as outlined in (4.19).

$$t_s \in Q_y \tag{4.18}$$

$$Q_{ya} = Q_y \setminus \{t_s\} \tag{4.19}$$

This terminology is used because we are deciding which task should be assigned first (the selected task), while the affected tasks are those influenced by the selected task. Specifically, the selected task can influence the waiting time of the affected tasks, and a poor choice for this task could result in significant delays for the others. Our goal is to identify the selected task that results in the fewest affected tasks being dropped while keeping the waiting time for the remaining, non-dropped affected tasks as low as possible. After assigning the optimal selected task to the MEC, we calculate the updated availability times of the MECs to determine the new earliest MEC availability time. A new decision window is created, spanning from the start processing time of the assigned task (previous earliest availability time) to the newly determined earliest availability time. This window includes the non-dropped affected tasks as well as any tasks that arrive before the new earliest availability time. We repeat the process to determine the next optimal task for assignment, with only one task being assigned to a MEC at a time. Maximum waiting time, $t_i^{\text{w\_max}}$, as determined in (4.20), represents the longest period that task i can wait

before being processed while still meeting its deadline. The task is dropped if it exceeds this period.

$$t_i^{\text{w\_max}} = t_i^{\text{d}} - t_i^{\text{p}} - t_i^{\text{ar}} \quad \forall i \in Q_y \tag{4.20}$$

According to (4.21), the selected task's start processing time, t_s^{sp}, is equal to the earliest availability time of the MEC server, t_e^{av}, added to the algorithm's execution time, ϵ_c. The execution time is high in the PSO algorithm, whereas it is significantly lower in On-Dyn-CDA.

$$t_s^{\text{sp}} = t_e^{\text{av}} + \epsilon_c \tag{4.21}$$

The waiting time in window y for an affected task, t_a^{w}, is defined as the duration between an affected task's arrival time and the earliest availability time at the moment when a specific selected task is being evaluated. In other words, this waiting time is only used to determine the optimal selected task and does not represent the final waiting time. This implies that when evaluating a particular task as the selected task, we determine the waiting time it imposes on the affected tasks. This method helps assess how choosing a suboptimal selected task could lead to longer waiting times for subsequent tasks. As defined in (4.22), d_{sa} is a binary variable that indicates whether an affected task is dropped as a result of selecting a specific task to be assigned to the MEC. If the sum of the selected task's processing time, t_{sj}^{p}, and the waiting time in window y, t_a^{w}, of the affected task is less than or equal to the maximum waiting time, $t_a^{\text{w\_max}}$, the affected task will not be dropped. Otherwise, it is dropped due to the impact of the selected task.

$$d_{sa} = \begin{cases} 0 & \text{if } t_s^{\text{p}} + t_a^{\text{w}} \leq t_a^{\text{w\_max}} \\ 1 & \text{otherwise} \end{cases} \tag{4.22}$$

To dynamically approximate the global objective defined in (4.10), which jointly minimizes end-to-end latency and the number of dropped tasks across the entire task set, the On-Dyn-CDA algorithm employs a local heuristic objective given in (4.23).

$$\arg\min_{s} \left\{ \sum_{\substack{a=1 \\ a \neq s}}^{W_y} \left(\frac{t_s^{\text{p}} + t_a^{\text{w}}}{t_a^{\text{w\_max}}} \cdot (1 - d_{sa}) + d_{sa} \right) \right\} \tag{4.23}$$

This equation serves as a tractable, local approximation that guides dynamic decision-making at each step of the task assignment process. Specifically, (4.23) is used to evaluate

the impact of selecting a particular task on the waiting time and drop likelihood of other tasks in the decision window.

Algorithm 1: On-Dyn-CDA: Online Dynamic Cost-Driven Assignment

---

Initialization:
Assign first two tasks to MECs and compute t_j^{av} for each.
while unassigned tasks remain **do**
    Compute $t_e^{av} = \arg\min_j(t_j^{av})$
    $Q_y \leftarrow \{t_i \mid t_i^{ar} \leq t_e^{av}, \text{meets deadline}\}$
    for each $t_s \in Q_y$ **do**
        $cost_s \leftarrow 0$, $Q_{ya} \leftarrow Q_y \setminus \{t_s\}$
        for each $t_a \in Q_{ya}$ **do**
            Compute t_a^w, $t_a^{w\_max}$
            if $t_s^p + t_a^w \leq t_a^{w\_max}$ **then**
                $cost_s$ += $\frac{t_s^p + t_a^w}{t_a^{w\_max}}$
            else
                $cost_s$ += 1
            end if
        end for
    end for
    Select $t_{\text{opt}} = \arg\min(cost_s)$
    Assign t_{opt}, update t_j^{av}
    Update window with remaining and new tasks
end while

By iteratively applying this heuristic in a greedy fashion, always selecting the task that minimizes this local cost, the On-Dyn-CDA algorithm approximates good solutions to the global problem without incurring its computational overhead. This means that, to determine the best selected task for assignment to the MEC, we seek the one that minimizes the number of dropped tasks and reduces the waiting impact on those that are not dropped. If d_{sa} is 1, there is no need to check the impact of the selected task on the waiting time of the affected task. However, if it is 0, this impact must be evaluated as the ratio of the sum of the selected task's processing time, t_{sj}^{p}, and the affected task's waiting time in window y, t_a^{w}, to the maximum waiting time, $t_a^{\text{w\_max}}$. This ratio can be referred to as the waiting cost since it measures the relative delay experienced by an affected task. The value of d_{sa} is

limited to either 0 or 1, whereas the waiting cost ranges between 0 and 1. The formulation in (4.23) is effective because it balances two key aspects of task scheduling: feasibility (whether a task is dropped) and timeliness (how close a task is to exceeding its deadline). The binary term d_{sa} penalizes cases where the affected task a would be dropped due to the selection of task s, directly addressing the feasibility constraint. For non-dropped tasks ($d_{sa} = 0$), the cost is expressed as a normalized ratio which measures the portion of the affected task's maximum waiting time that is consumed when task s is selected. A lower ratio implies that the affected task remains well within its allowable waiting time, minimizing its risk of deadline violation. In this way, the expression effectively quantifies the waiting cost experienced by each affected task. By combining the drop penalty, d_{sa}, and the relative waiting cost, the objective in (4.23) prioritizes task assignments that not only avoid drops but also preserve temporal flexibility for future scheduling decisions. This makes it a computationally efficient yet practical approximation of the global objective in (4.10).

4.3.3 Algorithm

Algorithm 1 outlines the On-Dyn-CDA method for real-time task offloading in MEC-enabled environments. The algorithm works by repeatedly selecting the most efficient task to assign to the MEC server that will become available next. At each step, it forms a decision window containing tasks that have arrived before the earliest MEC availability time. The task with the lowest calculated cost, which reflects both delay and drop rate, is selected and assigned. After the assignment, the MEC availability is updated and a new decision window is constructed. This process continues until all tasks are processed or dropped.

4.4 Performance Evaluation

4.4.1 Experimental Setup

In this study, we simulate a highway environment using the SUMO platform, where an RSU facilitates computational offloading for vehicles traveling along the highway. The RSU is equipped with two MEC servers to process incoming tasks generated by vehicles within its coverage area. Each task has a time-sensitive deadline, determined by the Euclidean distance between the vehicle and the RSU at the time of offloading. The simulation

examines three traffic density scenarios to evaluate the performance of the MEC servers under varying workloads.

Each vehicle generates a single image processing task, with task generation following a Poisson distribution to capture the randomness and variability typical of real-world vehicular offloading. Tasks can be generated either while a vehicle is within the RSU's range or before entering it, creating a more realistic simulation. The size of each task depends on the resolution of the image being processed [91]. To model different computational loads, we simulate varying image resolutions, with the data size of each image calculated based on a color depth of 24 bits per pixel, following standard RGB encoding. The parameters have been set based on [77], [24], as presented in Table 4.3.

Table 4.3: Simulation Parameters

Parameters	**Value**
The number of vehicles	50, 100, 200
The number of tasks per vehicle	1
The number of MEC servers	2
The number of CPUs per MEC server	1
Task size	2160, 3840, 6000, 8640 kb
Max bandwidth	20 Mhz
Noise power	100 dbm
Transmit power	200 mW

Table 4.4: PSO Parameters

Parameters	**Value**
Swarm Size	50
Maximum number of iterations	100
Cognitive Coefficient	1.49
Social Coefficient	1.49

The system operates with a maximum bandwidth of 20 MHz; however, accounting for a 4.6% guard band, the effective usable bandwidth is 19.08 MHz [1]. Simulations are performed on a machine with a Core i7 CPU, GeForce MX150 GPU, and 8GB of RAM. The parameters used in the PSO algorithms [62] are detailed in Table 4.4. These parameters are selected based on empirical observations from multiple experiments.

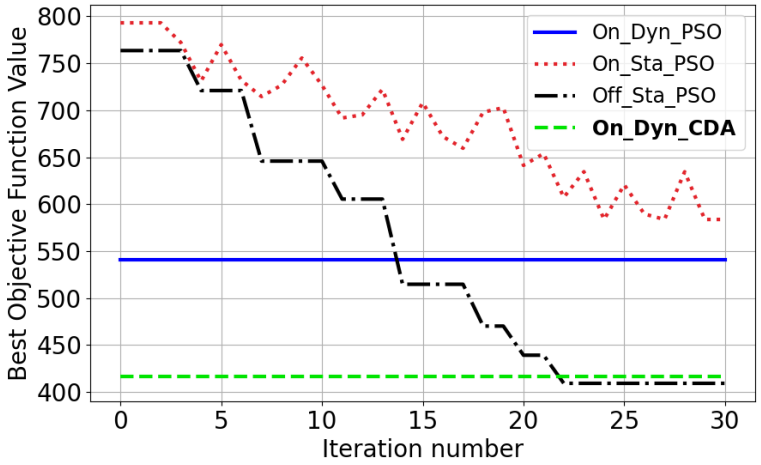


Figure 4.4: Comparison of Convergence Plots for different algorithms with 200 tasks

4.4.2 Numerical Results

FCFS and SDF are employed as deterministic methods to demonstrate how simple approaches perform in addressing a complex task offloading problem. PSO is applied in various configurations, including different simulation types (online and offline) and task processing strategies (static and dynamic), to evaluate the performance of a well-known optimization algorithm in diverse environments. Finally, we introduce a novel algorithm, On-Dyn-CDA, designed to overcome the inefficiency of traditional optimization algorithms in dynamic scenarios. This algorithm achieves results in terms of the number of dropped tasks and E2E latency that are close to the theoretical limit (Off-Sta-PSO) but with significantly reduced algorithm execution time. This highlights that On-Dyn-CDA not only meets the primary objectives but also processes a large number of tasks quickly, making it highly suitable for real-time task offloading. In the Numerical Results section, Table 4.5 presents the numerical performance results for each method.

Convergence

In Figure 4.4, we plot the objective function values over the iterations for 200 tasks. The convergence pattern of On-Sta-PSO, as shown in the figure, demonstrates its difficulty in finding the global optimum. This is primarily due to the significant waiting time introduced in this approach, which makes it challenging for the algorithm to explore the solution space effectively. Since processing does not begin until all tasks have arrived and PSO has

been executed, the extended delays lead to inefficiencies, causing the algorithm to struggle in balancing task assignments and minimizing the objective function. These constraints explain the slower convergence and the tendency to get stuck in suboptimal solutions.

Off-Sta-PSO demonstrates good convergence as it does not account for the major waiting times. It successfully finds the global optimum, achieving the best possible value. For On-Dyn-PSO, which is executed multiple times on different batches of tasks, the objective function value is calculated at the end based on the final total order. The same approach is applied for On-Dyn-CDA. The results indicate that the objective value of On-Dyn-CDA is better than that of On-Dyn-PSO, primarily because of the significant execution time required by PSO.

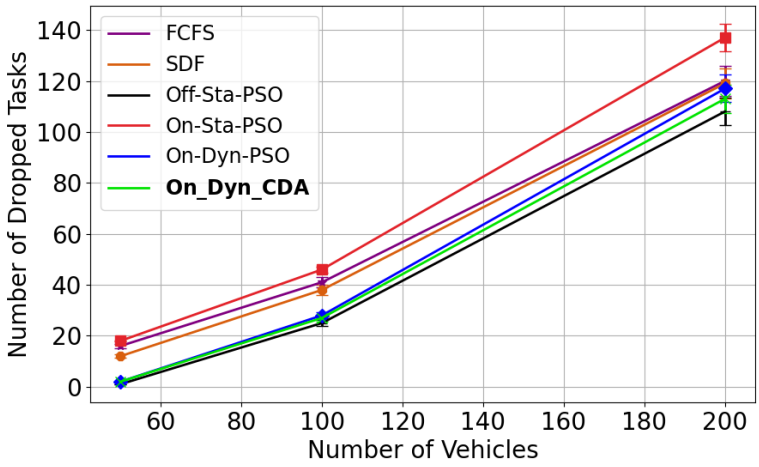


Figure 4.5: Number of Dropped Tasks for different numbers of vehicles across various algorithms

Dropped Task Ratio

Figure 4.5 shows the number of dropped tasks for different vehicle counts, including confidence intervals, across various algorithms over 10 runs. This indicates that as the number of tasks increases, the number of dropped tasks also rises across all algorithms due to longer waiting times. Off-Sta-PSO achieves the best performance, while On-Sta-PSO performs the worst. On-Dyn-CDA delivers results very close to the theoretical limit of Off-Sta-PSO, demonstrating strong performance. On-Dyn-PSO performs worse than On-Dyn-CDA but still achieves better results than FCFS and SDF. Despite the dynamic nature of On-Dyn-PSO, its performance is affected by the considerable execution time required by PSO to

process different batches of tasks. SDF performs better than FCFS by reducing the number of dropped tasks. However, as the number of tasks increases, the results of SDF and FCFS converge. This happens because the complexity of the task processing environment increases with a larger number of computationally intensive tasks, making SDF less efficient in maintaining its advantage over FCFS.

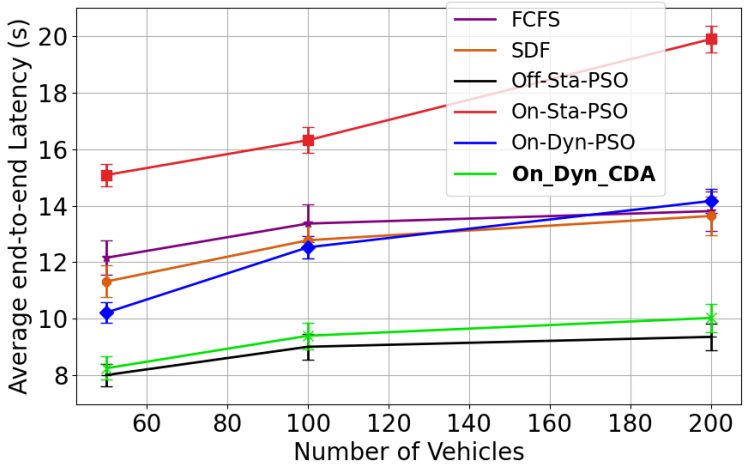


Figure 4.6: Average end-to-end latency for different numbers of vehicles across various algorithms

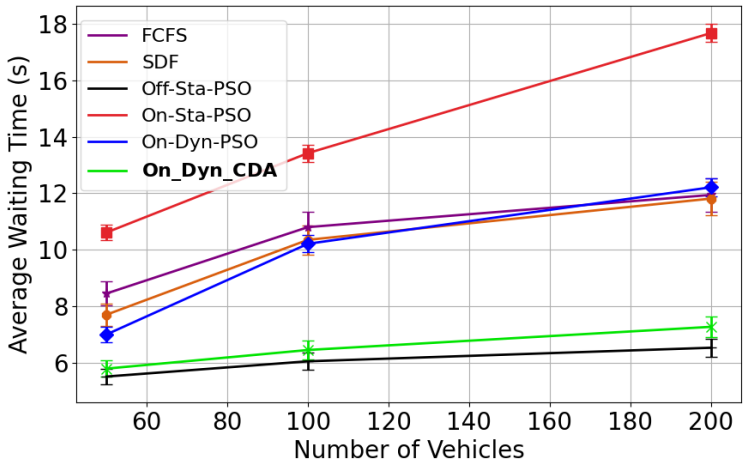


Figure 4.7: Average waiting time for different numbers of vehicles across various algorithms

Computation Latency

The comparison of average E2E latency is presented in Figure 4.6 for different vehicle counts, including confidence intervals, across various algorithms over 10 runs. Among the evaluated approaches, Off-Sta-PSO and On-Dyn-CDA demonstrate the best performance, with Off-Sta-PSO representing the theoretical optimum. On-Dyn-PSO initially achieves a lower average E2E latency compared to FCFS and SDF. However, as the number of tasks increases, the execution time of On-Dyn-PSO plays a major role in increasing the overall latency, leading to a decline in its performance compared to FCFS and SDF. Notably, On-Sta-PSO consistently performs the worst, exhibiting the highest average E2E latency across all scenarios. Figure 4.7 presents the average waiting time for different vehicle counts, including confidence intervals, across various algorithms over 10 runs. As shown, the average waiting time results are similar to those observed for the average E2E latency, since waiting time serves as the primary contributing factor to E2E latency. In Figure 4.8 the execution time for all algorithms is displayed over 10 runs, indicating the overall duration required for each algorithm to process all tasks, with standard deviation used to reflect the variability across runs. As observed, FCFS, SDF, and On-Dyn-CDA exhibit the shortest algorithm execution time, whereas the PSO-based models require significantly more time. As expected, On-Sta-PSO demonstrates the longest execution time, followed by On-Dyn-PSO, with Off-Sta-PSO performing better but still requiring more time compared to the other approaches. On-Dyn-PSO requires more execution time than Off-Sta-PSO because it performs real-time optimizations, continuously updating task assignments and server selections based on changing system conditions. In contrast, Off-Sta-PSO precomputes schedules offline under static conditions, avoiding the need for continuous updates and reducing computational overhead.

Table 4.5: Overall performance of all methods

Algorithm	Total end-end Latency (s)			Total waiting time (s)			Algorithm Execution Time (s)		
	50 Vehicles	100 Vehicles	200 Vehicles	50 Vehicles	100 Vehicles	200 Vehicles	50 Vehicles	100 Vehicles	200 Vehicles
FCFS	413.44	789.02	1104.78	278.32	637.25	955.01	0.0052	0.0078	0.02
SDF	430.03	792.12	1105.18	292.57	642.05	956.32	0.0075	0.0117	0.0313
Off-Sta-PSO	392.49	675.75	861.32	270.21	453.61	600.84	210.61	800.02	1300.02
On-Sta-PSO	483.10	881.28	1253.80	339.42	724.65	1112.91	213.42	805.10	1308.24
On-Dyn-PSO	460.12	827.12	1175.78	314.93	673.7	1013.33	217.54	821.30	1330.05
On-Dyn-CDA	**396**	**686.2**	**872.62**	**278.11**	**471.12**	**632.12**	**0.02**	**0.03**	**0.05**

For better comparison, Figure 4.9 presents the execution time of only the non-PSO algorithms. As illustrated, FCFS has the shortest execution time, followed by SDF, and finally On-Dyn-CDA. Table 4.5 presents the numerical results for total E2E Latency, total

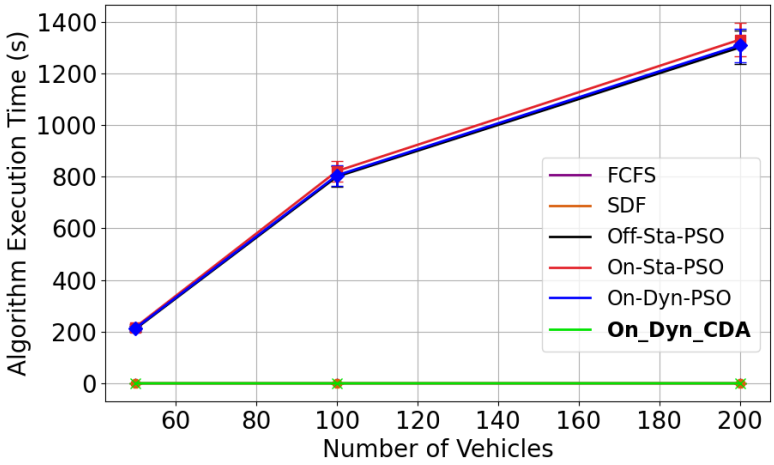


Figure 4.8: Algorithm execution time for different number of users under different algorithms

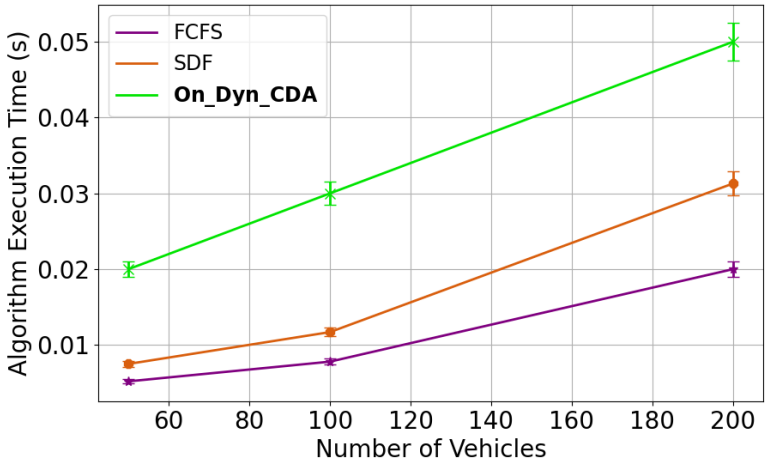


Figure 4.9: Algorithm execution time for different numbers of vehicles for FCFS, SDF, and On-Dyn-CDA

waiting time, and algorithm execution time in seconds. These results represent the standard deviation over 10 runs, using the same simulation setup for all algorithms to ensure consistency and fairness in comparison. As shown in Figure 4.8 and the numerical values in Table 4.5 for algorithm execution time, the execution time for On-Dyn-CDA does not increase significantly with the number of vehicles, unlike PSO-based algorithms. This

demonstrates the scalability of the On-Dyn-CDA algorithm.

4.5 Conclusion

Our comparative study of task offloading algorithms highlights the trade-offs between simplicity, optimization, and real-time performance in dynamic computational environments. FCFS and SDF, while straightforward, struggle to maintain efficiency as task complexity increases. PSO, when applied in offline and online configurations, demonstrates varying strengths and limitations. The offline static approach achieves the theoretical optimum, offering the best possible performance in terms of dropped tasks and latency. However, it lacks adaptability to dynamic scenarios, making it impractical for real-world applications. On-Dyn-PSO, while dynamic, suffers from a large execution time that limits its effectiveness in fast-changing environments. The proposed On-Dyn-CDA algorithm addresses this gap by achieving 95.58% of the theoretical limit and outperforming On-Dyn-PSO by 3.42% in terms of dropped tasks, as evaluated on a total of 200 tasks. It also achieves 93.32% closeness to the theoretical limit for average latency, with a 29.22% improvement compared to On-Dyn-PSO, based on a total of 200 tasks. On-Dyn-CDA significantly outperforms both PSO-based models and ML models. It demonstrates a faster execution time than PSO in the dynamic approach and eliminates the need for training and datasets. Its superior adaptability and simplicity demonstrate that it is an efficient and scalable solution for addressing dynamic computational demands. In future work, we plan to focus on energy optimization and the incorporation of task dependencies to further improve the practicality and efficiency of the task offloading process. Additionally, we will explore the potential of ML models, such as RL, and compare their performance with On-Dyn-CDA to identify optimal solutions for complex and dynamic offloading environments.

Acknowledgment

This work was supported in part by funding from the IDEaS program from the DND, in part by NSERC CREATE TRAVERSAL Program, and in part by the NSERC DISCOVERY Program.

Chapter 5

Meeting Deadlines in Motion: Deep RL for Real-Time Task Offloading in Vehicular Edge Networks

VEC drives the future by enabling low-latency, high-efficiency data processing at the very edge of vehicular networks. This drives innovation in key areas such as autonomous driving, intelligent transportation systems, and real-time analytics. Despite its potential, VEC faces significant challenges, particularly in adhering to strict task offloading deadlines, as vehicles remain within the coverage area of the RSU for only brief periods. To tackle this challenge, this chapter evaluates the performance boundaries of task processing by initially establishing a theoretical limit using PSO in a static environment. To address more dynamic and practical scenarios, PSO, DQN, and PPO models are implemented in an online setting. The objective is to minimize dropped tasks and reduce E2E latency, covering both communication and computation delays. Experimental results demonstrate that the DQN model considerably surpasses the dynamic PSO approach, achieving a substantial reduction in execution time. Furthermore, It leads to a reduction in dropped tasks by 2.5% relative to dynamic PSO and achieves 18.6% lower E2E latency, highlighting the effectiveness of DRL in enabling scalable and efficient task management for VEC systems.

5.1 Introduction

MEC significantly enhances vehicular task offloading by enabling vehicles to delegate complex computations to nearby edge servers, rather than relying solely on onboard systems or

distant cloud infrastructure [19]. By supporting real-time data analysis, MEC helps vehicles operate more intelligently within their surroundings. This integration of edge computing with connected vehicles forms the foundation for Ubiquitous Intelligence, where smart decision-making happens seamlessly across the network, contributing to safer and more adaptive transportation ecosystems. Despite the advantages MEC brings to vehicular task offloading, several challenges remain, particularly in meeting the strict latency and deadline constraints of vehicle-based applications. One major issue is ensuring that offloaded tasks are not only processed quickly but also returned to the vehicle before it moves out of the coverage range of a RSU [70]. Since vehicles are highly mobile and only stay connected to an RSU for a short period, any delay in processing or communication can result in missed deadlines and dropped tasks. Reducing task drop rates is therefore critical. One approach is to use optimization techniques such as PSO [70]. These methods can help determine the most efficient order in which tasks should be assigned and processed by edge servers, aiming to minimize task drop rates [60], [63]. However, a key limitation is that such optimization algorithms often require noticeable computational time, making them less suitable for real-time scenarios where rapid decisions are critical. In particular, the dynamic version of PSO tends to cause a higher task drop rate, primarily due to its high execution time. In contrast, ML models offer a promising alternative for real-time task offloading. Once trained, ML models can quickly predict optimal or near-optimal task assignments. Building on the advantages of ML, RL can be even more effective [26]. Unlike traditional ML models, RL focuses on learning optimal policies for sequential decision-making, rather than just predicting outcomes. It can improve upon suboptimal behaviors in the data by reasoning about long-term effects of actions, making it ideal for complex tasks where decisions impact future outcomes. This work makes the following key contributions:

1. We establish a theoretical limit for task offloading performance in vehicular networks by implementing PSO in a static and ideal environment.

2. We present a dynamic environment that utilizes a decision window mechanism to manage incoming tasks prior to MEC availability. The proposed approach prioritizes reducing the number of dropped tasks while minimizing the E2E latency. Decision-making within the window relies on a reward function in RL models and on dynamic PSO.

3. A novel dynamic reward function is employed during the training of DQN and PPO. The resulting best policy is then tested on a new dataset, eliminating the need for retraining across different datasets.

In the following sections, Section II reviews related work, Section III presents the pro-

Table 5.1: Assessment of our work in relation to existing research

Paper	Theoritical Limit	Dynamic	MTG	Training		KPI			Algorithm
				Off	On	AET	Latency	DTN	
[111]	✗	✓	-	-	-	✗	✓	✗	PSO, Integer Constraint Relaxation Iterative (ICRI)
[22]	✗	✓	MATLAB	-	-	✗	✓	✗	Optimal Joint Task offloading and Resource allocation algorithm (OJTR), Heuristic Joint Task offloading and Resource allocation algorithm (HJTR)
[71]	✗	✓	Python	-	✓	✗	✓	✗	Deep Meta-RL, DQN
[7]	✗	✓	-	✓	-	✗	✓	✗	AODAI, ACTO-n, DQN, SARSA
[16]	✗	✓	DAGGEN	✓	-	✓	✓	✗	SMRL-MTO, Policy Gradient, PPO, DQN
Ours	✓	✓	SUMO	✓	-	✓	✓	✓	Dynamic PSO, DQN, PPO

** MTG: Mobility Trace Generator, Off: Offline, On: Online, KPI: Key Performance Indicator, AET: Algorithm Execution Time, DTR: Drop Task Ratio*

posed methodology, Section IV discusses the performance evaluation, and Section V provides the conclusion.

5.2 Related Work

Extensive research has been conducted on task offloading in MEC to address its inherent limitations and enhance efficiency. However, a significant challenge persists in meeting the rapidly increasing demand for ultra-low-latency task processing in IoV environments. To tackle this, optimization techniques such as PSO have been explored. In one approach, PSO is used to jointly determine assistant vehicle selection and transmit power in a hybrid MEC-V2X setting, where tasks can be offloaded not only to RSUs but also to nearby vehicles [111].

In addition to optimization methods with high execution times, it's crucial to account for faster approaches. A recent study tackles this challenge by combining optimization techniques such as generalized benders decomposition and a low-complexity heuristic algorithm to efficiently minimize task processing delays [22]. RL has also emerged as a powerful tool for task offloading in MEC due to its ability to make dynamic decisions in complex

environments. Building on this, an online Deep Meta-RL framework models the offloading process as Markov Decision Processes (MDPs) to enhance adaptability in B5G/6G-enabled IoV systems and introduces a two-loop learning approach for fast decision-making [71]. A recent approach introduces two novel RL-based methods, a value-based AODAI and a policy-based ACTO-n scheme [7].

Furthermore, a Seq2Seq-based Meta-RL framework can be developed to tackle MTO in dynamic environments. By framing the offloading process as a series of MDPs and integrating a Bi-GRU encoder with attention mechanisms and a model-agnostic meta-learning (MAML) approach, this method enables rapid adaptation to changing task structures and MEC configurations [16]. While these studies provide valuable insights, many do not provide comparisons to a theoretical limit and the reporting of execution times, both of which are critical for real-time decision-making. Another important consideration often overlooked is the number of dropped tasks and the objective of minimizing them. Table 5.1 compares our approach with existing studies, highlighting the gaps that this chapter addresses.

Our work builds upon our previous research [70], extending it to dynamic environments. We deploy PSO, DQN, and PPO within a dynamic framework, demonstrating that RL models not only achieve significantly lower execution times compared to PSO, but also deliver performance close to the theoretical limit.

5.3 Methodologies under Study

5.3.1 The Offloading Schemes

Based on Figure 5.1, we first introduce Off-Sta-PSO, an offline and static PSO variant running on SUMO-generated mobility traces, representing a theoretical limit in an ideal environment. Tasks are handled offline with prior knowledge, excluding execution time. To explore dynamic scenarios, we implement On-Dyn-PSO, applying PSO online for real-time task offloading without prior knowledge. Due to PSO's long execution time, it is unsuitable for real-time applications, leading us to adopt DQN and PPO for off-policy and on-policy RL solutions. In dynamic settings, tasks arriving before the earliest MEC availability form a queue called decision window where the model assigns tasks to minimize dropped tasks and E2E latency. If a MEC is already available, tasks are assigned immediately, reducing execution complexity. DQN and PPO models are first trained on randomized SUMO traces (Baseline model), then tested on the target dataset without retraining (Test-only model) to ensure generalization, robustness, and efficient deployment.

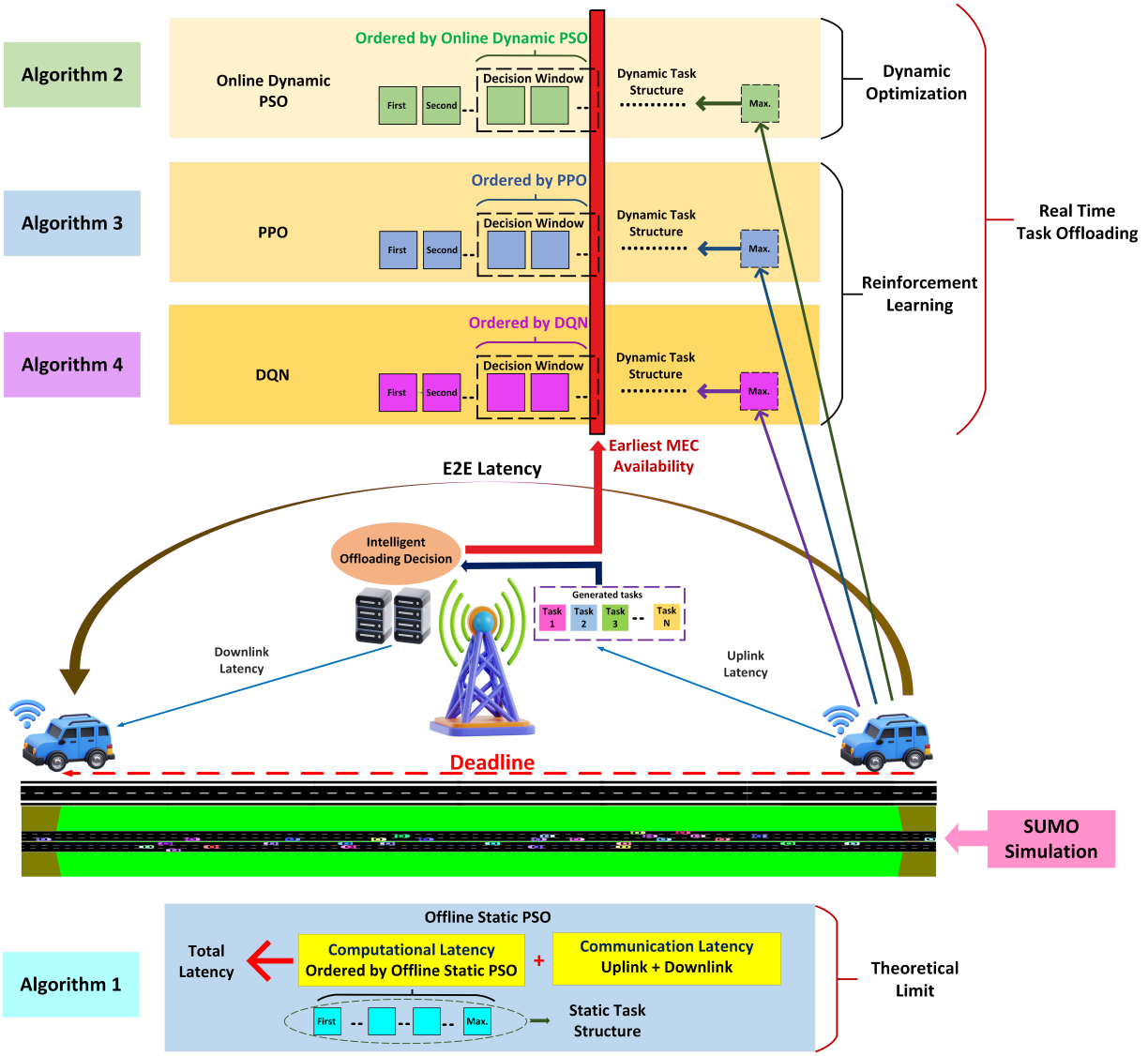


Figure 5.1: Overview of Vehicular Edge Computing for Static and Dynamic Task Offloading

5.3.2 The mathematical framework for task offloading

Table 5.2: Notation table

Parameter	Description
T_i^{sp}	Start processing time for task i
T_i^{p}	Processing time for task i
T_i^{w}	Waiting time for task i
T_i^{ar}	Arrival time of task i on the RSU
L_i^{p}	Computation Latency of task i
S_i	Size of task i
r_i	transmission rate for task i
T_i^{cm}	Communication time for task i
L_i^{e2e}	End-to-end latency for task i
x_{ij}	Binary decision variable for assigning task i to MEC j
D	Number of dropped tasks
N	Total number of tasks
M	Total number of MEC servers
T_j^{av}	The time when MEC j becomes available
T_e^{av}	Earliest MEC availability time
T_{yl}^{ar}	Arrival time of task l
T_{yl}^{D}	The deadline of task l in subset y
T_{yl}^{R}	Remaining time in the RSU range for task l in subset y
T_{yl}^{p}	Processing time required for task l
$g_{y,l}^{\mathrm{d}}$	Time gap related to task l in subset y
$R_{y,l}^{\mathrm{L}}$	Latency-based reward for assigning task l in subset y
$R_{y,l}^{\mathrm{d}}$	Drop-task reward for assigning task l in subset y
$R_{y,l}^{\mathrm{T}}$	Total reward for assigning task l in subset y
T	The complete collection of tasks in the simulation
S	The collection of tasks within a decision window
Q_y	The set of tasks within a decision window that can be processed and returned before their deadlines

(Table 5.2 continues on next page)

Parameter	Description
s_t	The state of the system at time t
a_y	The action taken at subset y
$\mathcal{A}(y)$	The complete set of possible actions for subset y
$P(y)$	The processing time for all tasks within subset y

Computation Latency

Since we have M MEC servers, the first M tasks are assigned to a server and begin processing as soon as they arrive at the RSU, denoted by T_i^{ar}, while the start processing time T_i^{sp} for the remaining tasks is set to the moment their preceding task has finished processing. Furthermore, the task processing times T_i^{p} correspond to the inference times reported in [91]. The notations employed in the mathematical formulas are detailed in Table 5.2. Most tasks experience a waiting time T_i^{w} at the RSU before being processed. Based on (5.1), the computation latency L_i^{p} for each task is the sum of its processing time and waiting time.

$$L_i^{\text{p}} = T_i^{\text{p}} + \underbrace{(T_i^{\text{sp}} - T_i^{\text{ar}})}_{T_i^{\text{w}}} \tag{5.1}$$

Communication Latency

During transmission, if multiple tasks within the RSU's range are offloaded simultaneously, the available bandwidth is shared among them. In such cases, the bandwidth allocated to each task is proportional to its size [70]. However, if a task is transmitted individually, it can utilize the entire available bandwidth. Based on our study in [70], sharing the bandwidth can slightly reduce transmission latency compared to using a fixed bandwidth allocation. The set of concurrent tasks is defined as:

$$\mathcal{N} \;=\; \left\{\, i = 1, \ldots, N' \;\middle|\; \mathcal{T}_i = \mathcal{T} \right\} \tag{5.2}$$

$\mathcal{T}_i$ denotes the offloading time when task i becomes ready for transmission, while $\mathcal{T}$ represents the offloading time at which all tasks in $\mathcal{N}$ are ready to be offloaded. Additionally, N' represents the total number of tasks in set $\mathcal{N}$. The bandwidth assigned to task i is calculated using (5.3). This bandwidth is further used to calculate the transmission rate, r_i, which is needed to determine the transmission time.

$$B_i = \begin{cases} B_{\max} \cdot \frac{S_i}{\sum_{i=1}^{N'} S_i} & N' > 1 \\ B_{\max} & N' = 1 \end{cases} \tag{5.3}$$

Where S_i represents the size of task i. The data rate, r_i, is defined in (5.4), with p indicating the transmission power, g referring to the channel gain, and n_0 denoting the noise power density.

$$r_i = B_i \cdot \log_2 \left(1 + \frac{p \cdot g}{n_0}\right) \tag{5.4}$$

To calculate the communication time T_i^{cm} we require this transmission rate r_i. We assume that downlink communication is identical to uplink communication. As defined in (5.5), the E2E latency is the sum of the computation latency and both communication times T_i^{cm}.

$$L_i^{\text{e2e}} = L_i^{\text{p}} + (2 \times \underbrace{\frac{S_i}{r_i}}_{T_i^{\text{cm}}}) \tag{5.5}$$

Dropped Tasks

A task is assigned to a MEC server only if its E2E latency does not exceed its deadline. If a task cannot meet this condition, it will be dropped. This assignment is represented by a binary variable x_{ij} as illustrated in (5.6).

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to MEC } j \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

Optimization Problem

Our goal is to minimize dropped tasks D first, then reduce E2E latency for remaining tasks. The objective function (5.7) balances these factors, with λ controlling the emphasis on latency and $(1 - \lambda)$ prioritizing dropped task reduction. λ is set empirically through iterative testing to favor D.

$$\min \left(\lambda \left(\sum_{j=1}^{M} \sum_{i=1}^{N} L_i^{\text{e2e}} \cdot x_{ij} \right) + (1-\lambda) \underbrace{\left(\frac{1}{N} \sum_{i=1}^{N} \left(1 - \sum_{j=1}^{M} x_{ij} \right) \right)}_{D} \right) \tag{5.7}$$

According to constraint (5.8), each task can be assigned to and processed by only one MEC server.

$$\sum_{j=1}^{M} x_{ij} \leq 1 \quad \forall i \tag{5.8}$$

Real-Time Approach

To process tasks in real time, it is necessary to determine the availability of MEC servers after each assignment, which is computed according to (5.9).

$$T_j^{\text{av}} = (T_i^{\text{sp}} + T_i^{\text{p}}) \cdot x_{ij} \quad \forall j \tag{5.9}$$

Let $T = \{t_1, t_2, \ldots, t_N\}$ represent the set of all tasks generated within the simulation. Once each task is assigned to a MEC server and all servers' availability times are updated, a decision window is formed. This decision window consists of all tasks that are waiting to be offloaded and is represented as $S = \{t_{i_1}, t_{i_2}, \ldots, t_{i_W}\}$ where W is the total number of tasks in S, while $S \subseteq T$. A subset of tasks Q_y within the decision window S is considered for assignment. These are tasks that arrive before the earliest availability and are not assigned yet. The subset Q_y is defined mathematically as:

$$Q_y = \left\{ t_{y_l} \in S \;\middle|\; 1 \leq l \leq W_y,\ T_{y_l}^a \leq T_e^{\text{av}},\ T_{y_l}^w \leq T_{y_l}^R - T_{y_l}^p - T_{y_l}^{\text{cm}} \right\},$$

where t_{y_l} is the task at position l in window y.

Reinforcement Learning

The definitions of state and action spaces in the RL model are as follows:

State Space It represents the current status of the system, which includes:

- $\{T_j^{\text{av}}\}_{j=1}^{M}$: Availability times for all M MEC servers.
- $\{T_{y_l}^{\text{ar}}, T_{y_l}^{\text{D}}, T_{y_l}^{p}\}_{y=1}^{W_s}, {}_{l=1}^{W_y}$: Characteristics of tasks in the set of Q_y.

At time t, the system state is represented as:

$$s_t = \left\{ \{T_j^{\text{av}}\}_{j=1}^{M}, \{T_{y_l}^{\text{ar}}, T_{y_l}^{\text{R}}, T_{y_l}^{p}\}_{y=1}^{W_s}, {}_{l=1}^{W_y} \right\}.$$

Action Space The action space defines the possible assignments for tasks in a subset y. The action a_y selects which specific task from the subset Q_y should be offloaded. Mathematically, this is represented as:

$$a_y \in \{1, 2, \ldots, W_y\}, \quad a_y = l \iff \text{assign task } t_{y_l},$$

The full action space, denoted as $\mathcal{A}(y)$, is given by:

$$\mathcal{A}(y) = \{1, 2, \ldots, W_y\},$$

where W_y represents the total number of tasks in the subset Q_y.

Reward Function The reward mechanism in the RL model is composed of two complementary components, designed to balance two objectives, latency and dopped tasks.

Drop-Task Reward To encourage timely task assignment, a *time gap* $g_{y,l}^{d}$ is calculated for each task t_{y_l} in the subset y. This is based on the difference between the task's deadline and the MEC server's earliest availability time:

$$g_{y,l}^{\text{d}} = T_{y_l}^{\text{D}} - T_e^{\text{av}}, \quad \forall y, l.$$

The total time gap for all tasks in the subset y, denoted G_y^{d}, is computed as:

$$G_y^{\text{d}} = \sum_{l=1}^{W_y} g_{y,l}^{\text{d}}, \quad \forall y.$$

Finally, the reward for assigning a specific task t_{y_l} at window y is defined as:

$$R^{\mathrm{d}}_{y,l} = \frac{100}{G^{\mathrm{d}}_{y}} \cdot \left(G^{\mathrm{d}}_{y} - \widetilde{g^{\mathrm{d}}_{y,l}}\right), \quad \forall y, l,$$

where $\widetilde{g^{\mathrm{d}}_{y,l}}$ represents the time gap associated with the task selected by the RL algorithm. This formulation ensures that tasks with tighter deadlines relative to the MEC's availability are assigned a higher priority.

Latency-Based Reward The cumulative processing time for all tasks in the subset y, denoted as L_y, is then:

$$P_y = \sum_{l=1}^{W_y} T^{p}_{y_l}, \quad \forall y,$$

The reward for assigning a specific task t_{y_l} at window y is formulated to encourage the minimization of latency:

$$R^{\mathrm{L}}_{y,l} = \frac{100}{P_y} \cdot \left(P_y - \widetilde{T^{P}_{y_l}}\right), \quad \forall y, l,$$

This formulation ensures that tasks with lower computational latencies are prioritized, promoting efficiency in the assignment process. $R^{\mathrm{T}}_{y,l}$, the total reward for assigning t_{y_l} at window y, is formulated as (5.10).

$$R^{\mathrm{T}}_{y,l} = R^{\mathrm{d}}_{y,l} + R^{\mathrm{L}}_{y,l} \quad \forall\, y, l \tag{5.10}$$

5.4 Performance Analysis

5.4.1 Experimental Setup

SUMO generates mobility traces with two MEC servers embedded on an RSU. Task deadlines are derived from Euclidean distance at offloading, arrivals follow a Poisson distribution, and task size depends on image resolution [91]. Workloads of 50, 100, and 200 tasks simulate varying traffic conditions. Simulation, PSO, and RL model parameters are configured based on [40, 62, 69, 70, 105], summarized in Table 5.3.

Table 5.3: Parameters values

Parameters	**Value**
Max bandwidth	20 Mhz
λ	0.4
Personal & global learning coefficients	1.49
DQN learning rate	0.0001
PPO learning rates of actor and critic	0.0003
DQN discount factor	0.9
PPO discount factor	0.95

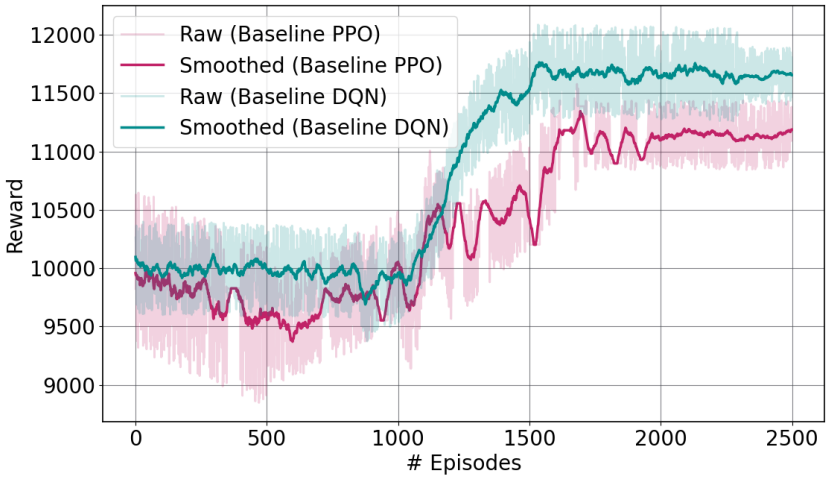


Figure 5.2: Reward Values for RL models

5.4.2 Numerical Results

Figure 5.2 illustrates the reward values for DQN and PPO over 2500 training episodes. Both DQN and PPO begin with rewards near 10,000, but PPO initially declines while DQN remains steady. After 1000 episodes, DQN progresses more rapidly, achieving higher rewards than PPO. Overall, DQN delivers stronger and more consistent performance. Moreover, Figure 5.3 shows the optimization objective function values for Test-only both DQN and PPO, alongside other algorithms for comparison. In a dynamic setting, only the final sum of the best objective values from each run is reported. Similarly, since Test-only RL models are deployed for a single episode, their objective values remain constant. DQN demonstrates stronger performance, achieving an objective value closest to that of Off-Sta-

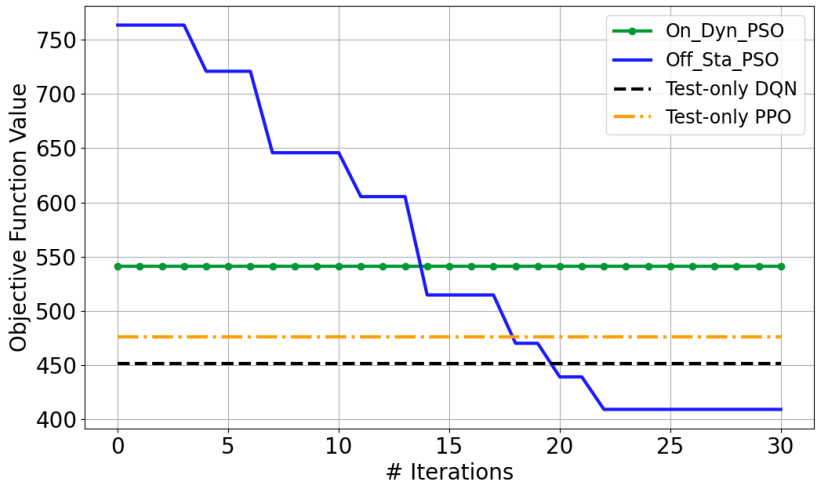


Figure 5.3: Objective Function Values for all algorithms

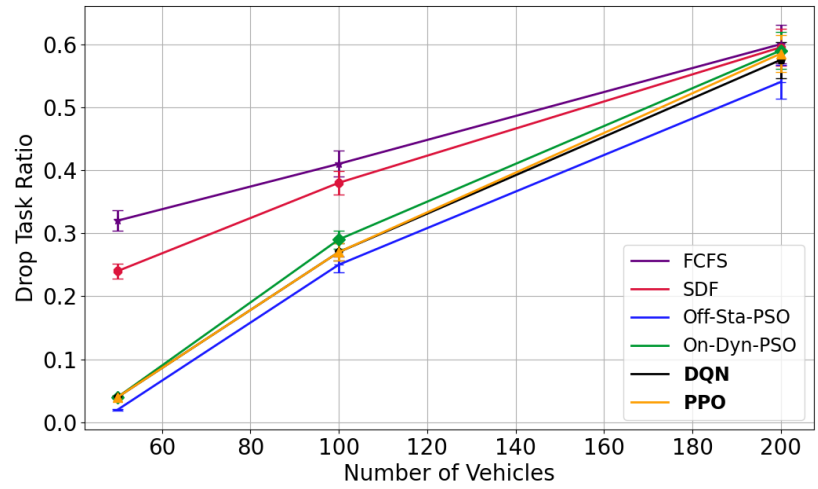


Figure 5.4: Average Drop Task Ratio

PSO (theoretical limit). All key performance metric values are averaged across 10 runs. The variation across runs indicates that even slight differences in decision-making can result in different tasks being dropped or processed, leading to significant changes in E2E latency and waiting time. Figure 5.4 presents the drop task ratio for all algorithms under varying numbers of vehicles. The results show that Test-only DQN performs close to the theoretical limit and outperforms both On-Dyn-PSO and PPO. Similarly, the average E2E

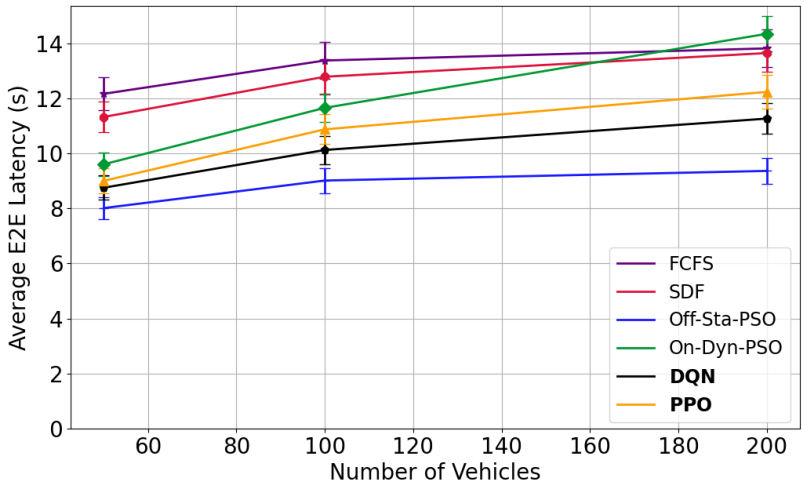


Figure 5.5: Average E2E Latency

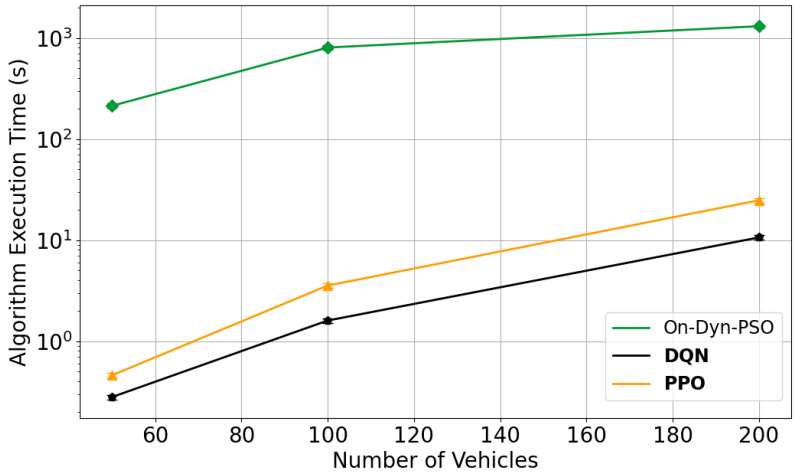


Figure 5.6: Logarithmic Algorithm Execution Time

latency results are shown in Figure 5.5. This indicates that Test-only DQN continues to outperform On-Dyn-PSO, which struggles to maintain performance under a high number of vehicles. This is primarily due to the high execution time of On-Dyn-PSO. Compared to PPO, DQN results in lower E2E latency as well. Due to the small magnitude of the original execution times, their logarithmic values are presented in Figure 5.6 for more effective comparison. The values show that RL methods are faster than Dynamic PSO,

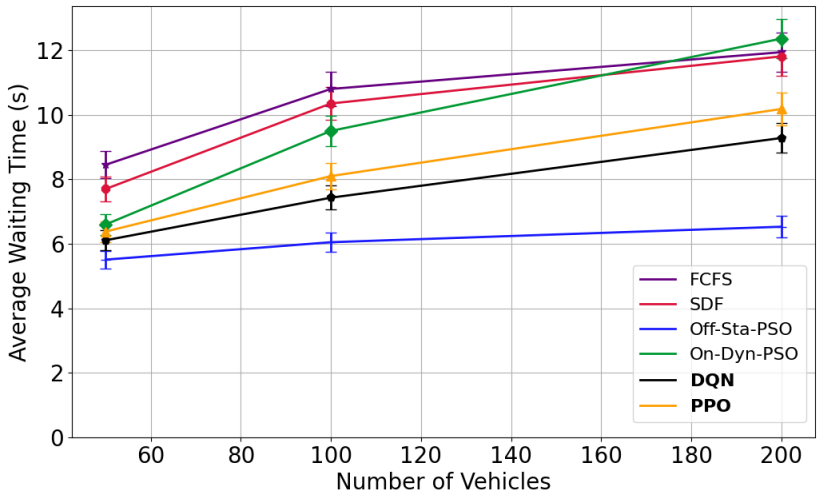


Figure 5.7: Average Waiting Time

which reduces the extra waiting time added due to algorithm execution, and as a result, leads to fewer dropped tasks. Among the RL approaches, DQN runs faster than PPO and also achieves better overall performance, making it the best trade-off between speed and effectiveness. For better model comparison, average waiting times are shown in Figure 5.7. These results are also reported for FCFS and SDF, serving as benchmarks. Execution time details for Test-only PPO and DQN models are found in Table 5.4.

Table 5.4: Total and Per-Window Execution Times for Test-only DQN and PPO

Algorithm	Total Execution Time			Number of Decision Windows			Average Execution Time (s)		
	50 Vehicles	100 Vehicles	200 Vehicles	50 Vehicles	100 Vehicles	200 Vehicles	50 Vehicles	100 Vehicles	200 Vehicles
Test-only DQN	0.28	1.6	10.62	20	62	178	0.014	0.026	0.06
Test-only PPO	0.46	3.55	24.78	20	64	177	0.023	0.056	0.14

5.5 Conclusion

This study tackles real-time task offloading in vehicular networks by evaluating heuristic and RL models. Key contributions include establishing a theoretical limit with static PSO, proposing a dynamic decision window to reduce dropped tasks and latency, and training DQN and PPO with dynamic rewards for policy transfer without retraining. The proposed DQN model significantly outperforms the On-Dyn-PSO, achieving a significant reduction

in execution time, as well as 18.6% lower E2E latency and 2.5% fewer dropped tasks. Notably, the DQN model achieves performance that is closest to the optimal benchmark in terms of both E2E latency and the drop task ratio. Compared to PPO, DQN offers a 57.1% decrease in execution time, as well as 5.7% lower E2E latency and 1.7% fewer dropped tasks. To ensure the reliability of results, we conduct multiple independent simulation runs and report averaged outcomes with standard deviations. Our RL models are evaluated in a separate "Test-only" phase on unseen mobility traces to confirm generalization. All models are tested under identical task and network conditions, and inference is performed with deterministic policies, eliminating variance from exploration. These practices collectively ensure statistical validity, consistency, and real-world relevance of the results. DQN outperforms PPO because it handles discrete, window-based task selection more efficiently, while PPO is better suited for continuous action spaces and requires more samples to stabilize. In future work, we plan to consider deploying additional models that require no training and offer faster execution, and explore task partitioning techniques using RL methods.

Acknowledgment

This work was supported in part by funding from the IDEaS program from the DND, in part by NSERC CREATE TRAVERSAL Program, in part by Ontario Research Fund-Research Excellence (ORF-RE) program under RE012-026, and in part by the NSERC DISCOVERY Program.

Chapter 6

Conclusion and Future Directions

VEC has emerged as a promising paradigm to meet the computational and latency requirements of connected vehicles by enabling offloading of intensive tasks to nearby edge servers or peer vehicles. However, achieving efficient, scalable, and real-time task offloading in VEC environments remains a significant challenge due to dynamic task arrival rates, limited resources, variable network conditions, and the need for fast decision-making. This thesis is motivated by the critical need to optimize task offloading strategies in VEC to ensure low latency, high reliability, and efficient resource utilization. The primary problem addressed in this research is how to design and evaluate intelligent task offloading mechanisms that can adapt to dynamic and heterogeneous vehicular environments without compromising on performance or scalability. To address this problem, the following key research questions are posed:

- How can task partitioning and bandwidth sharing improve the efficiency of task offloading in VEC systems?
- What are the trade-offs between deterministic, metaheuristic, and cost-based offloading algorithms in terms of performance, adaptability, and execution time?
- Can RL be effectively applied to real-time task offloading, and how does it compare to traditional approaches under dynamic conditions?

Through the design and evaluation of various models and algorithms across three core chapters, this thesis has systematically explored these questions. The following sections summarize the chapter-wise contributions, highlight the key findings, and discuss future directions for advancing research in this domain.

6.1 Concluding Highlights

In our research, we explicitly account for the dynamic topology and association patterns characteristic of VEC by incorporating mobility-aware scheduling and offloading strategies. Rather than just assuming static or persistent connections, we model task offloading and resource allocation with respect to frequently changing links between vehicles and RSUs. This includes considering factors such as vehicle speed, predicted trajectory, and link duration when making scheduling decisions. Furthermore, we evaluate algorithms in real-time settings to examine how well they adapt to changes in connectivity and association. This is done by introducing a decision window mechanism, which adapts to the real-time and evolving characteristics of tasks as they are generated by the simulation application. By doing so, our work reflects the transient and location-sensitive nature of VEC, ensuring that proposed solutions remain effective under realistic mobility and network dynamics.

Chapter 3 demonstrates that both task partitioning and bandwidth sharing can enhance VEC performance. In particular, guaranteed task completion is achieved through task partitioning, where tasks are divided into subtasks that are intelligently distributed between local execution on vehicles and offloading to edge servers. This collaborative processing approach improves resource utilization, reduces latency, and ensures reliable task completion. Notably, the partitioning approach significantly outperforms MEC-only scenarios, eliminating task drops in a 200-task simulation, where the MEC-only setup drops 109 tasks. While shared bandwidth slightly improved task handling over fixed bandwidth, partitioning proved more effective in reducing both dropped tasks and average E2E latency.

Chapter 4 compares deterministic-based (FCFS, SDF) and metaheuristic-based (PSO) offloading methods under both static and dynamic settings. It reveals that while FCFS and SDF are simple, they falter under complex task loads. Offline PSO reaches theoretical optimums but lacks real-time applicability. The proposed On-Dyn-CDA algorithm bridges this gap by achieving 95.58% of the optimal performance in dropped tasks and 93.32% in latency, while outperforming On-Dyn-PSO by 3.42% in terms of reduced task drops and 29.22% in latency. It also achieves much faster runtime performance than On-Dyn-PSO and avoids the need for training, proving its scalability and efficiency.

Focusing on real-time task offloading, Chapter 5 explores RL models with adaptive reward structures. Specifically, it applies DQN and PPO algorithms to enable online scheduling in highly dynamic vehicular environments. These models learn to make offloading decisions based on changing network conditions, task deadlines, and vehicle mobility, aiming to maximize deadline satisfaction while minimizing latency. By continuously adapting to environmental changes, the RL-based approach supports robust and efficient task management in motion. The DQN model achieves remarkable gains, with a considerable

reduction in execution time and improved latency and task drop rates compared to On-Dyn-PSO. Against PPO, DQN reduces latency by 5.7%, drops by 1.7%, and execution time by 57.1%. DQN's discrete action handling and efficient sample use make it better suited for VEC scenarios than PPO.

This research shows that optimization methods like PSO are effective in static or offline settings, where task arrivals and system states are fully known in advance. In such cases, they can provide near-optimal scheduling decisions by exploring a wide solution space. However, due to their high computational cost and long execution times, they are less suitable for real-time vehicular environments. In contrast, RL approaches like DQN demonstrate strong performance in dynamic and time-sensitive scenarios. Once trained, RL models can make fast, adaptive decisions in response to changing network conditions and vehicle mobility, making them better suited for real-time task offloading in VEC.

6.2 Future Directions and Open Issues

Building on the findings of this thesis, several promising directions can be pursued to enhance VEC systems further. Below, we outline future work and open challenges.

Future research will focus on incorporating energy-aware offloading strategies to reduce power consumption, which is especially important in resource-constrained VEC environments. Beyond energy optimization, adopting a multi-objective approach that combines energy consumption, latency, and task drop rate into a unified objective function will help better reflect real-world trade-offs.

While the proposed On-Dyn-CDA algorithm demonstrates strong performance, there are several areas for further improvement. One key direction is to address task dependencies, as the current model assumes that all tasks are independent. Including inter-task relationships would make the scheduling process more realistic and complex. Finally, scalability testing on larger and more diverse vehicular networks will be crucial for validating the model's effectiveness in more demanding environments.

While the DQN model shows strong performance, it still requires an initial training phase. Future research could explore training-free or lightweight alternatives, such as rule-based RL or few-shot learning methods, to reduce training time and computational costs. Another promising direction is extending RL techniques beyond task selection to include task partitioning, enabling more comprehensive and intelligent offloading solutions. Additionally, hybrid methods that combine RL with metaheuristic techniques, such as a DQN combined with PSO, may yield better performance and adaptability.

References

[1] 3GPP TS 38.101-1. 5g; nr; user equipment (ue) radio transmission and reception. Technical Specification (TS) 38.213, ETSI, 10 2022. Version 17.6.0.

[2] Manzoor Ahmed, Salman Raza, Muhammad Ayzed Mirza, Abdul Aziz, Manzoor Ahmed Khan, Wali Ullah Khan, Jianbo Li, and Zhu Han. A survey on vehicular task offloading: Classification, issues, and challenges. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4135–4162, 2022.

[3] Zhengyang Ai, Weiting Zhang, Mingyan Li, Pengxiao Li, and Lei Shi. A smart collaborative framework for dynamic multi-task offloading in iiot-mec networks. *Peer-to-Peer Networking and Applications*, 16(2):749–764, 2023.

[4] Annu and P. Rajalakshmi. Towards 6g v2x sidelink: Survey of resource allocation—mathematical formulations, challenges, and proposed solutions. *IEEE Open Journal of Vehicular Technology*, 5:344–383, 2024.

[5] N. Anupriya, R. Venkateswari, and N. Iswarya. Intelligent resource management in heterogeneous vehicular networks through deep reinforcement learning. In *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, pages 1–6, 2024.

[6] Muhammed Nur Avcil, Mujdat Soyturk, and Burak Kantarci. Fair and efficient resource allocation via vehicle-edge cooperation in 5g-v2x networks. *Vehicular Communications*, page 100773, 2024.

[7] Ta Huu Binh, Do Bao Son, Hiep Vo, Binh Minh Nguyen, and Huynh Thi Thanh Binh. Reinforcement learning for optimizing delay-sensitive task offloading in vehicular edge–cloud computing. *IEEE Internet of Things Journal*, 11(2):2058–2069, 2024.

[8] Arash Bozorgchenani, Setareh Maghsudi, Daniele Tarchi, and Ekram Hossain. Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions. *IEEE Transactions on Mobile Computing*, 21(12):4233–4248, 2022.

[9] Chaofan Chen, Wendi Nie, Yaoxin Duan, Victor C.S. Lee, Kai Liu, and Huamin Li. An adaptive data rate-based task offloading scheme in vehicular networks. In *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, pages 877–884, 2022.

[10] Haiming Chen, Wei Qin, and Lei Wang. Task partitioning and offloading in iot cloud-edge collaborative computing framework: a survey. *Journal of Cloud Computing*, 11(1):86, 2022.

[11] Shuaijie Chen, Wenfeng Li, Jingtao Sun, Pasquale Pace, Lijun He, and Giancarlo Fortino. An efficient collaborative task offloading approach based on multi-objective algorithm in mec-assisted vehicular networks. *IEEE Transactions on Vehicular Technology*, pages 1–14, 2025.

[12] Xiao Chen. Federated reinforcement learning-driven offloading strategy for edge computing. In *2024 6th International Conference on Electronics and Communication, Network and Computer Technology (ECNCT)*, pages 467–470, 2024.

[13] Ying Chen, Fengjun Zhao, Xin Chen, and Yuan Wu. Efficient multi-vehicle task offloading for mobile edge computing in 6g networks. *IEEE Transactions on Vehicular Technology*, 71(5):4584–4595, 2022.

[14] Zheyi Chen, Zhiqin Huang, Junjie Zhang, Hongju Cheng, and Jie Li. Resource allocation and collaborative offloading in multi-uav-assisted iov with federated deep reinforcement learning. *IEEE Internet of Things Journal*, 12(5):4629–4640, 2025.

[15] Joseph Clancy, Darragh Mullins, Brian Deegan, Jonathan Horgan, Enda Ward, Ciarán Eising, Patrick Denny, Edward Jones, and Martin Glavin. Wireless access for v2x communications: Research, challenges and opportunities. *IEEE Communications Surveys Tutorials*, 26(3):2082–2119, 2024.

[16] Penglin Dai, Yaorong Huang, Kaiwen Hu, Xiao Wu, Huanlai Xing, and Zhaofei Yu. Meta reinforcement learning for multi-task offloading in vehicular edge computing. *IEEE Transactions on Mobile Computing*, 23(3):2123–2138, 2024.

[17] Wenfeng Dai. Joint task offloading, resource allocation and data caching in mec-assisted vehicular network. In *2023 4th International Conference on Computer Engineering and Application (ICCEA)*, pages 70–76, 2023.

[18] Alisson Barbosa de Souza, Paulo Antonio Leal Rego, Vinay Chamola, Tiago Carneiro, Paulo Henrique Gonçalves Rocha, and José Neuman de Souza. A bee colony-based algorithm for task offloading in vehicular edge computing. *IEEE Systems Journal*, 17(3):4165–4176, 2023.

[19] Hamza Djigal, Jia Xu, Linfeng Liu, and Yan Zhang. Machine and deep learning for resource allocation in multi-access edge computing: A survey. *IEEE Communications Surveys Tutorials*, 24(4):2449–2494, 2022.

[20] Jianbo Du, Yan Sun, Ning Zhang, Zehui Xiong, Aijing Sun, and Zhiguo Ding. Cost-effective task offloading in noma-enabled vehicular mobile edge computing. *IEEE Systems Journal*, 17(1):928–939, 2023.

[21] Qingyang Fan, Weizhe Zhang, Chen Ling, Rahul Yadav, Desheng Wang, and Hui He. Mobility-aware cooperative service caching for mobile augmented reality services in mobile edge computing. *IEEE Transactions on Vehicular Technology*, 73(11):17543–17557, 2024.

[22] Wenhao Fan, Yi Su, Jie Liu, Shenmeng Li, Wei Huang, Fan Wu, and Yuan'an Liu. Joint task offloading and resource allocation for vehicular edge computing based on v2i and v2v modes. *IEEE Transactions on Intelligent Transportation Systems*, 24(4):4277–4292, 2023.

[23] Mohamed Amine Ferrag, Othmane Friha, Burak Kantarci, Norbert Tihanyi, Lucas Cordeiro, Merouane Debbah, Djallel Hamouda, Muna Al-Hawawreh, and Kim-Kwang Raymond Choo. Edge learning for 6g-enabled internet of things: A comprehensive survey of vulnerabilities, datasets, and defenses. *IEEE Communications Surveys & Tutorials*, 25(4):2654–2713, 2023.

[24] Mingjin Gao, Rujing Shen, Long Shi, Wen Qi, Jun Li, and Yonghui Li. Task partitioning and offloading in dnn-task enabled mobile edge computing networks. *IEEE Transactions on Mobile Computing*, 22(4):2435–2445, 2021.

[25] Anousheh Gholami and John S. Baras. Collaborative cloud-edge-local computation offloading for multi-component applications. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 361–365, 2021.

[26] Sambi Reddy Gottam and Udit Narayana Kar. Power controlled resource allocation and task offloading via optimized deep reinforcement learning in d2d assisted mobile edge computing. *IEEE Access*, 13:19420–19437, 2025.

[27] Shidrokh Goudarzi, Seyed Ahmad Soleymani, Mohammad Hossein Anisi, Anish Jindal, and Pei Xiao. Optimizing uav-assisted vehicular edge computing with age of information: An sac-based solution. *IEEE Internet of Things Journal*, 12(5):4555–4569, 2025.

[28] Lina A. Haibeh, Mustapha C. E. Yagoub, and Abdallah Jarray. A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches. *IEEE Access*, 10:27591–27610, 2022.

[29] Xuewen He, Yuhao Cen, Yinsheng Liao, Xin Chen, and Chao Yang. Optimal task offloading strategy for vehicular networks in mixed coverage scenarios. *Applied Sciences*, 14(23), 2024.

[30] Ling Hou, Mark A. Gregory, and Shuo Li. A survey of multi-access edge computing and vehicular networking. *IEEE Access*, 10:123436–123451, 2022.

[31] Han Hu, Dingguo Wu, Fuhui Zhou, Shi Jin, and Rose Qingyang Hu. Dynamic task offloading in mec-enabled iot networks: A hybrid ddpg-d3qn approach. In *2021 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2021.

[32] Chongjing Huang, Qi Fu, Chaoliang Wang, and Zhaohui Li. Joint task offloading and scheduling algorithm in vehicular edge computing networks. In *2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 318–323, 2023.

[33] Jing Huang, Wenyu Wu, Weihong Huang, Yufeng Xiao, Li Lisi, and Jinxi Sun. A survey on task partitioning and scheduling for vehicular edge computing. In *2023 IEEE 10th International Conference on Cyber Security and Cloud Computing (CSCloud)/2023 IEEE 9th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 336–342, 2023.

[34] Xiujie Huang, Yuhao Chen, Zhiquan Liu, Shancheng Zhao, Zhetao Li, Renzhang Chen, and Quanlong Guan. Task offloading based on the fusion of model-and data-driven intelligence for vehicular edge computing networks. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–16, 2025.

[35] Asmaa Ibrahim and Bassem Mokhtar. What if vec is moving: Probabilistic model of task execution through offloading in vehicular computing environments. *IEEE Access*, 12:170889–170897, 2024.

[36] Akhirul Islam, Arindam Debnath, Manojit Ghose, and Suchetana Chakraborty. A survey on task offloading in multi-access edge computing. *Journal of Systems Architecture*, 118:102225, 2021.

[37] Suyun Kang, Fanghe Lu, Wanming Hao, and Shouyi Yang. Resource allocation and offloading strategy in mobile edge computing considering mobility and inter-user relevance. In *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pages 1–5, 2022.

[38] Jaehwan Lee and Woongsoo Na. A survey on vehicular edge computing architectures. In *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, pages 2198–2200, 2022.

[39] Fanyuan Li, Yan Lin, Nuoheng Peng, and Yijin Zhang. Deep reinforcement learning based computing offloading for mec-assisted heterogeneous vehicular networks. In *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, pages 927–932, 2020.

[40] Han Li, Ke Xiong, Yuping Lu, Wei Chen, Pingyi Fan, and Khaled Ben Letaief. Collaborative task offloading and resource allocation in small-cell mec: A multi-agent ppo-based scheme. *IEEE Transactions on Mobile Computing*, 24(3):2346–2359, 2025.

[41] Li Li and Pingzhi Fan. Latency and task loss probability for noma assisted mec in mobility-aware vehicular networks. *IEEE Transactions on Vehicular Technology*, 72(5):6891–6895, 2023.

[42] Mushu Li, Jie Gao, Ning Zhang, Lian Zhao, and Xuemin Shen. Collaborative computing in vehicular networks: A deep reinforcement learning approach. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[43] Song Li, Weibin Sun, Qiang Ni, and Yanjing Sun. Road side unit-assisted learning-based partial task offloading for vehicular edge computing system. *IEEE Transactions on Vehicular Technology*, 73(4):5546–5555, 2024.

[44] Yangqianhang Li and Li Li. Computation offloading and resource allocation in mec-enabled vehicular networks: Partial offloading versus binary offloading. In *2024 7th World Conference on Computing and Communication Technologies (WCCCT)*, pages 260–266, 2024.

[45] Ducsun Lim and Inwhee Joe. A drl-based task offloading scheme for server decision-making in multi-access edge computing. *Electronics*, 12(18):3882, 2023.

[46] Haoqiang Liu, Wenzheng Huang, Dong In Kim, Sumei Sun, Yonghong Zeng, and Shaohan Feng. Towards efficient task offloading with dependency guarantees in vehicular edge networks through distributed deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 73(9):13665–13681, 2024.

[47] Jinshi Liu, Manzoor Ahmed, Muhammad Ayzed Mirza, Wali Ullah Khan, Dianlei Xu, Jianbo Li, Abdul Aziz, and Zhu Han. Rl/drl meets vehicular task offloading using edge and vehicular cloudlet: A survey. *IEEE Internet of Things Journal*, 9(11):8315–8338, 2022.

[48] Lei Liu, Ming Zhao, Miao Yu, Mian Ahmad Jan, Dapeng Lan, and Amirhosein Taherkordi. Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks. *IEEE Transactions on Intelligent Transportation Systems*, 24(2):2169–2182, 2023.

[49] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[50] Yanfei Lu, Dengyu Han, Xiaoxuan Wang, and Qinghe Gao. Distributed task offloading for large-scale vec systems: A multi-agent deep reinforcement learning method. In *2022 14th International Conference on Communication Software and Networks (ICCSN)*, pages 161–165, 2022.

[51] Quyuan Luo, Changle Li, Tom H. Luan, and Weisong Shi. Minimizing the delay and cost of computation offloading for vehicular edge computing. *IEEE Transactions on Services Computing*, 15(5):2897–2909, 2022.

[52] Vallisha M, Vishal Khot, Sharan S Pai, Vineeth R Rao, and Kayarvizhy N. Deep reinforcement learning for task offloading in a multi-access edge computing environment. In *2023 International Conference on Network, Multimedia and Information Technology (NMITCON)*, pages 1–6, 2023.

[53] Qianpiao Ma, Hongli Xu, Haibo Wang, Yang Xu, Qingmin Jia, and Chunming Qiao. Fully distributed task offloading in vehicular edge computing. *IEEE Transactions on Vehicular Technology*, 73(4):5630–5646, 2024.

[54] Homa Maleki, Mehmet Başaran, and Lütfiye Durak-Ata. Handover-enabled dynamic computation offloading for vehicular edge computing networks. *IEEE Transactions on Vehicular Technology*, 72(7):9394–9405, 2023.

[55] Ali Matin and Hussein Dia. Impacts of connected and automated vehicles on road safety and efficiency: A systematic literature review. *IEEE Transactions on Intelligent Transportation Systems*, 24(3):2705–2736, 2023.

[56] Zhixin Mei, Hebing Du, Pan He, Aofei Dong, Kuiyuan Feng, and Jinkun Xu. Joint optimization of task offloading and resource allocation in heterogeneous edge networks. In *2024 6th International Conference on Data-driven Optimization of Complex Systems (DOCS)*, pages 601–606, 2024.

[57] Rodolfo Meneguette, Robson De Grande, Jo Ueyama, Geraldo P Rocha Filho, and Edmundo Madeira. Vehicular edge computing: Architecture, resource management, security, and challenges. *ACM Computing Surveys (CSUR)*, 55(1):1–46, 2021.

[58] Muhammad Ayzed Mirza, Yu Junsheng, Salman Raza, Manzoor Ahmed, Muhammad Asif, Azeem Irshad, and Neeraj Kumar. Mcla task offloading framework for 5g-nr-v2x-based heterogeneous vecns. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):14329–14346, 2023.

[59] Sungwon Moon and Yujin Lim. Task partitioning for migration with collaborative edge computing in vehicular networks. In *2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 102–107, 2021.

[60] Parisa Fard Moshiri, Murat Simsek, and Burak Kantarci. On the interplay between network metrics and performance of mobile edge offloading. In *ICC 2024 - IEEE International Conference on Communications*, pages 4018–4023, 2024.

[61] Parisa Fard Moshiri, Murat Simsek, and Burak Kantarci. Towards sustainable edge computing: Efficient task offloading for energy efficiency and latency reduction. In *GLOBECOM 2024-2024 IEEE Global Communications Conference*, pages 577–582. IEEE, 2024.

[62] Parisa Fard Moshiri, Murat Simsek, and Burak Kantarci. Joint optimization of completion ratio and latency of offloaded tasks with multiple priority levels in 5g edge. *IEEE Transactions on Network and Service Management*, 2025.

[63] Parisa Fard Moshiri, Murat Simsek, and Burak Kantarci. Partitioned task offloading for low-latency and reliable task completion in 5g mec, 2025.

[64] Hussein T. Mouftah, Melike Erol-Kantarci, and Sameh Sorour. *Connected and Autonomous Vehicles in Smart Cities*. CRC Taylor and Francis, Boca Raton, Florida, USA, 2020.

[65] Suleman Munawar, Zaiwar Ali, Muhammad Waqas, Shanshan Tu, Syed Ali Hassan, and Ghulam Abbas. Cooperative computational offloading in mobile edge computing for vehicles: A model-based dnn approach. *IEEE Transactions on Vehicular Technology*, 72(3):3376–3391, 2023.

[66] Ahmadun Nabi, Tanmay Baidya, and Sangman Moh. Comprehensive survey on reinforcement learning-based task offloading techniques in aerial edge computing. *Internet of Things*, 2024.

[67] Abdenacer Naouri, Hangxing Wu, Nabil Abdelkader Nouri, Sahraoui Dhelim, and Huansheng Ning. A novel framework for mobile-edge computing by optimizing task offloading. *IEEE Internet of Things Journal*, 8(16):13065–13076, 2021.

[68] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Xiping Hu, Jiangchuan Liu, Lei Guo, Bin Hu, Ricky Y. K. Kwok, and Victor C. M. Leung. Partial computation offloading and adaptive task scheduling for 5g-enabled vehicular networks. *IEEE Transactions on Mobile Computing*, 21(4):1319–1333, 2022.

[69] Liwen Niu, Xianfu Chen, Ning Zhang, Yongdong Zhu, Rui Yin, Celimuge Wu, and Yangjie Cao. Multiagent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems. *IEEE Internet of Things Journal*, 10(12):10519–10531, 2023.

[70] Mahsa Paknejad, Parisa Fard Moshiri, Murat Simsek, Burak Kantarci, and Hussein T Mouftah. A reliable and efficient 5g vehicular mec: Guaranteed task completion with minimal latency. *arXiv preprint arXiv:2503.19320*, 2025.

[71] Priyadarshni Priyadarshni, Praveen Kumar, Shivani Tripathi, Akshun Pratap Dubey, and Rajiv Misra. Mec- assisted task offloading using meta-reinforcement learning for b5g/6g network. In *2024 IEEE International Conference on Big Data (BigData)*, pages 4309–4314, 2024.

[72] Xianke Qiang, Zheng Chang, Yun Hu, Lei Liu, and Timo Hämäläinen. Adaptive and parallel split federated learning in vehicular edge computing. *IEEE Internet of Things Journal*, 12(5):4591–4604, 2025.

[73] Baghiani Radouane, Guezouli Lyamine, Korichi Ahmed, and Barka Kamel. Scalable mobile computing: From cloud computing to mobile edge computing. In *Intl. Conf. on Networking, Information Sys. and Security: Envisage Intelligent Systems in 5g//6G-based Interconnected Digital Worlds*, pages 1–6, 2022.

[74] Salman Raza, Shangguang Wang, Manzoor Ahmed, Muhammad Rizwan Anwar, Muhammad Ayzed Mirza, and Wali Ullah Khan. Task offloading and resource allocation for iov using 5g nr-v2x communication. *IEEE Internet of Things Journal*, 9(13):10397–10410, 2022.

[75] Chunli Ren, Guoan Zhang, Xiaohui Gu, and Yue Li. Computing offloading in vehicular edge computing networks: Full or partial offloading? In *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, volume 6, pages 693–698, 2022.

[76] Rim Sayegh, Abdulhalim Dandoush, Hela Marouane, and Sahar Hoteit. Preliminary evaluation and optimization of task offloading and latency in vehicular edge computing. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pages 109–111, 2024.

[77] Winston K.G. Seah, Chung-Hau Lee, Ying-Dar Lin, and Yuan-Cheng Lai. Combined communication and computing resource scheduling in sliced 5g multi-access edge computing systems. *IEEE Transactions on Vehicular Technology*, 71(3):3144–3154, 2022.

[78] Qiaoqiao Shen, Bin-Jie Hu, and Enjun Xia. Dependency-aware task offloading and service caching in vehicular edge computing. *IEEE Transactions on Vehicular Technology*, 71(12):13182–13197, 2022.

[79] Rujing Shen, Mingjin Gao, Wei Li, and Yonghui Li. Dynamic task offloading in distributed vec networks: An exploration and exploitation assisted contract-theoretic approach. *IEEE Transactions on Vehicular Technology*, 73(4):5717–5729, 2024.

[80] Sunita Sunil Shinde, S Parameswari, S R Arun Raj, K Kannan, N. Rajesh, and Ajmeera Kiran. Optimizing offloading in mec-enabled vehicular networks using adaptive pso and v2v communication. In *2023 Seventh International Conference on Image Information Processing (ICIIP)*, pages 369–373, 2023.

[81] Swapnil Sadashiv Shinde and Daniele Tarchi. Collaborative reinforcement learning for multi-service internet of vehicles. *IEEE Internet of Things Journal*, 10(3):2589–2602, 2023.

[82] Gang Sun, Zhiying Wang, Hanyue Su, Hongfang Yu, Bo Lei, and Mohsen Guizani. Profit maximization of independent task offloading in mec-enabled 5g internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 25(11):16449–16461, 2024.

[83] Geng Sun, Jiayun Zhang, Zemin Sun, Long He, and Jiahui Li. Collaborative task offloading in vehicular edge computing networks. In *2022 IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, pages 592–598, 2022.

[84] Yilong Sun, Zhiyong Wu, Ke Meng, and Yunhui Zheng. Vehicular task offloading and job scheduling method based on cloud-edge computing. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):14651–14662, 2023.

[85] Zemin Sun, Geng Sun, Yanheng Liu, Jian Wang, and Dongpu Cao. Bargain-match: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks. *IEEE Transactions on Mobile Computing*, 23(2):1655–1673, 2024.

[86] Huijun Tang, Huaming Wu, Guanjin Qu, and Ruidong Li. Double deep q-network based dynamic framing offloading in vehicular edge computing. *IEEE Transactions on Network Science and Engineering*, 10(3):1297–1310, 2023.

[87] Ihsan Ullah, Hyun-Kyo Lim, Yeong-Jun Seok, Naeem Ul Islam, Chang-Hun Ji, and Youn-Hee Han. Graph-powered reinforcement learning for intelligent task offloading in vehicular networks. In *2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 337–342, 2024.

[88] Truong Van Truong and Anand Nayyar. System performance and optimization in noma mobile edge computing surveillance network using ga and pso. *Computer Networks*, 223:109575, 2023.

[89] Bingxin Wang, Lei Liu, and Jie Wang. Actor-critic based drl algorithm for task offloading performance optimization in vehicle edge computing. In *2023 International Wireless Communications and Mobile Computing (IWCMC)*, pages 93–98, 2023.

[90] Bingxin Wang, Lei Liu, and Jie Wang. Multi-agent deep reinforcement learning for task offloading in vehicle edge computing. In *2023 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pages 1–6, 2023.

[91] Haoxin Wang, BaekGyu Kim, Jiang Xie, and Zhu Han. Energy drain of the object detection processing pipeline for mobile devices: Analysis and implications. *IEEE Transactions on Green Communications and Networking*, 5(1):41–60, 2021.

[92] Junhua Wang, Kun Zhu, Bing Chen, and Zhu Han. Distributed clustering-based cooperative vehicular edge computing for real-time offloading requests. *IEEE Transactions on Vehicular Technology*, 71(1):653–669, 2022.

[93] Siyu Wang, Bo Yang, Zhiwen Yu, Xuelin Cao, Yan Zhang, and Chau Yuen. Computation offloading for multi-server multi-access edge vehicular networks: A ddqn-based method. In *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pages 1–5, 2024.

[94] Wenhua Wang, Yilin Zhang, Yutao Zhang, Qin Liu, Tian Wang, and Weijia Jia. Online dependent task offloading by application partitioning in edge intelligence for internet of vehicles. *IEEE Internet of Things Journal*, 12(5):4860–4871, 2025.

[95] Xin Wang, Jianhui Lv, Adam Slowik, Byung-Gyu Kim, B. D. Parameshachari, Keqin Li, and Gang Feng. Augmented intelligence of things for priority-aware task offloading in vehicular edge computing. *IEEE Internet of Things Journal*, 11(22):36002–36013, 2024.

[96] Ye Wang, Yanheng Liu, Zemin Sun, Lingling Liu, Jiahui Li, and Geng Sun. Priority-aware task offloading and resource allocation in vehicular edge computing networks. In *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*, pages 205–212, 2022.

[97] Zhiying Wang, Gang Sun, Hanyue Su, Hongfang Yu, Bo Lei, and Mohsen Guizani. Low-latency scheduling approach for dependent tasks in mec-enabled 5g vehicular networks. *IEEE Internet of Things Journal*, 11(4):6278–6289, 2024.

[98] Chunyi Wu, Lin Li, Li Zhang, Chao Gao, Xingchen Wu, and Shan Xiao. Efficient gan-based federated optimization for vehicular task offloading with mobile edge computing in 6g network. *IEEE Internet of Things Journal*, 12(3):2736–2748, 2025.

[99] Jiaqi Wu, Ming Tang, Changkun Jiang, Lin Gao, and Bin Cao. Cloud-edge–end collaborative task offloading in vehicular edge networks: A multilayer deep reinforcement learning approach. *IEEE Internet of Things Journal*, 11(22):36272–36290, 2024.

[100] Ke Xiao, Zhixin Mei, Aofei Dong, Kuiyuan Feng, and Penglin Dai. Joint task offloading and resource allocation in software-defined networking-enabled vehicular edge computing: A multi-objective approach. In *2024 IEEE International Symposium on Product Compliance Engineering - Asia (ISPCE-ASIA)*, pages 1–7, 2024.

[101] Tiyin Xiao, Yuanyuan Qi, Tao Shen, Yan Feng, and Lin Huang. Intelligent task offloading method for vehicular edge computing based on improved-sac. In *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, volume 5, pages 1720–1725, 2022.

[102] Zheng Xue, Chang Liu, Canliang Liao, Guojun Han, and Zhengguo Sheng. Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems. *IEEE Transactions on Vehicular Technology*, 72(5):6709–6722, 2023.

[103] Jinming Yang, Awais Aziz Shah, and Dimitrios Pezaros. A survey of energy optimization approaches for computational task offloading and resource allocation in mec networks. *Electronics*, 12(17):3548, 2023.

[104] Zhiyong Cui Yinhai Wang and Ruimin Ke. *Machine Learning for Transportation Research and Applications*. Elsevier, 2023.

[105] Zhuyue Yu, Yuliang Tang, Lintao Zhang, and Hao Zeng. Deep reinforcement learning based computing offloading decision and task scheduling in internet of vehicles. In *2021 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1166–1171, 2021.

[106] Bolei Zhang, Fu Xiao, and Lifa Wu. Offline reinforcement learning for asynchronous task offloading in mobile edge computing. *IEEE Transactions on Network and Service Management*, 21(1):939–952, 2024.

[107] Cui Zhang, Wenjun Zhang, Qiong Wu, Pingyi Fan, Qiang Fan, Jiangzhou Wang, and Khaled B. Letaief. Distributed deep reinforcement learning-based gradient quantization for federated learning enabled vehicle edge computing. *IEEE Internet of Things Journal*, 12(5):4899–4913, 2025.

[108] Jindie Zhang, Dongyang Yan, Xiaohong Jing, and Rongrong Qian. A sojourn time based algorithm for vehicular edge task offloading. In *2024 IEEE 24th International Conference on Communication Technology (ICCT)*, pages 605–610, 2024.

[109] Lushi Zhang, Hongzhi Guo, Xiaoyi Zhou, and Jiajia Liu. Trusted task offloading in vehicular edge computing networks: A reinforcement learning based solution. In *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, pages 6711–6716, 2023.

[110] Xinran Zhang, Jingyuan Liu, Tao Hu, Zheng Chang, Yanru Zhang, and Geyong Min. Federated learning-assisted vehicular edge computing: Architecture and research directions. *IEEE Vehicular Technology Mag.*, 18/4:75–84, 2023.

[111] Yilun Zhang, Changrun Chen, Huiling Zhu, Yijin Pan, and Jiangzhou Wang. Latency minimization for mec-v2x assisted autonomous vehicles task offloading. *IEEE Transactions on Vehicular Technology*, 74(3):4917–4932, 2025.

[112] Yilun Zhang, Changrun Chen, Huiling Zhu, and Jiangzhou Wang. Task offloading for mec-v2x assisted autonomous driving. In *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pages 01–05, 2024.

[113] Zheng Zhang and Feng Zeng. Efficient task allocation for computation offloading in vehicular edge computing. *IEEE Internet of Things Journal*, 10(6):5595–5606, 2023.

[114] Liang Zhao, Enchao Zhang, Shaohua Wan, Ammar Hawbani, Ahmed Y. Al-Dubai, Geyong Min, and Albert Y. Zomaya. Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing. *IEEE Transactions on Mobile Computing*, 23(5):4259–4272, 2024.

[115] Lei Zhu, Zhizhong Zhang, Lilan Liu, Linlin Feng, Peng Lin, and Yu Zhang. Online distributed learning-based load-aware heterogeneous vehicular edge computing. *IEEE Sensors Journal*, 23(15):17350–17365, 2023.

[116] Jinhui Zuo, Caixing Shao, Yang Yang, and Yin Ma. Task offloading optimization in vehicular edge computing with meta-policy models. In *2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM)*, pages 434–439, 2024.