Task Migration with Deadlines using Machine Learning-based Dwell Time Prediction in Vehicular Micro Clouds

Ziqi Zhou[a], Agon Memedi[a], Chunghan Lee[b], Seyhan Ucar[b], Onur Altintas[b], Falko Dressler[a]

[a]*School of Electrical Engineering and Computer Science, TU Berlin, Germany*
[b]*InfoTech Labs, Toyota Motor North America R&D, CA, U.S.A.*

Abstract

Edge computing is becoming ever more relevant to offload compute-heavy tasks in vehicular networks. In this context, the concept of vehicular micro clouds (VMCs) has been proposed to use compute and storage resources on nearby vehicles to complete computational tasks. As many tasks in this application domain are time critical, offloading to the cloud is prohibitive. Additionally, task deadlines have to be dealt with. This paper addresses two main challenges. First, we present a task migration algorithm supporting deadlines in vehicular edge computing. The algorithm is following the earliest deadline first model but in presence of dynamic processing resources, i.e., vehicles joining and leaving a VMC. This task offloading is very sensitive to the mobility of vehicles in a VMC, i.e., the so-called dwell time a vehicles spends in the VMC. Thus, secondly, we propose a machine learning-based solution for dwell time prediction. Our dwell time prediction model uses a random forest approach to estimate how long a vehicle will stay in a VMC. Our approach is evaluated using mobility traces of an artificial simple intersection scenario as well as of real urban traffic in cities of Luxembourg and Nagoya. Our proposed approach is able to realize low-delay and low-failure task migration in dynamic vehicular conditions, advancing the state of the art in vehicular edge computing.

Keywords: Edge computing, vehicular micro cloud, task migration, task offloading, dwell time prediction

1. Introduction

Vehicle-to-everything (V2X) networking technologies have been in the focus of our research community for a long time [1]. Now, with the development of cellular V2X (C-V2X) [2, 3] and mobile edge computing (MEC) in the context of 5G [4, 5], new opportunities for advanced communication and computation among modern vehicles arise. C-V2X enhances communication reliability and scalability, while MEC deploys computing resources close to vehicles. These technologies help make the idea of vehicular micro cloud (VMC) [6, 7] a reality. The concept of VMCs has been generalized in the form of virtualized edge computing [8], which is expected to be an integral part of 6G networks. In general, a VMC (or the virtualized edge) allows vehicles to utilize computing resources from other vehicles in close proximity. Migrating tasks within a VMC therefore helps decreasing task completion time cost and is particularly beneficial for delay-sensitive tasks. Compared to typical cloud-based technologies, VMCs serve as local edge servers, reducing communication delay. By clustering vehicles into small groups and leveraging the computational capacity of vehicles in the vicinity, VMCs enable more efficient resource allocation via task offloading.

Stationary and mobile vehicular micro clous are two types of VMCs, formed by a certain geographical region or moving cars respectively. This paper focuses on the stationary VMC where vehicles enter a predefined region and offload tasks to other vehicles only in this area. This collaborative computing approach can significantly reduce task completion times by utilizing currently idle high computing capacity resources from neighboring vehicles. However, the dynamic nature of vehicular environments brings several challenges to achieving this efficiently. One of these is the mobility of vehicles, which can lead to task failures due to migration problems due to imprecise information about near-future position, and thus connectivity, of the involved vehicles. When either the user or the vehicle currently processing the task leaves the VMC, the task cannot be completed.

In this paper, we present a heuristic for optimized task migration in a VMC. This extends our previous work in [9]. Our heuristic task migration strategy minimizes the task-finishing time. We propose a greedy migration algorithm that selects the least time-cost destination for each task to be offloaded. Making use of task deadlines, this resembles an earliest deadline first (EDF) behavior in a dynamic and mobile context. Our heuristic makes use of a vehicle dwell time prediction model to determine an approximate time the vehicle will still be part of the VMC. Specifically, we propose a machine learning (ML)-based dwell time prediction model using a random forest approach.

As an initial reference, we use an artificial single intersection to explore the performance of our heuristic. For proof of concept, we conducted simulations using realistic urban traffic data from the mid-scale European city Luxembourg [10], and the Japanese city Nagoya [11]. The task migration performance is evaluated using four metrics and a utility function that combines both task completion time and task failure rate, providing a view on practicability of the approach. It takes two types of task failure cases into account: i) cars leaving the VMC before task completion, and ii) task completion time exceeding the deadline.

This paper extends our previous work [9]. The main contributions beyond the current state of the art can be summarized as follows:

- We developed a novel task migration algorithm for use in vehicular micro clouds. The algorithm is based on a greedy approach, sequentially offloading tasks to the neighbor where it can be completed the earliest. In addition, deadlines are considered as a constraint. This matches the well-known EDF approach, now in a dynamic and mobile environment.
- In order to estimate the time a vehicle will be part of the VMC (the dwell time), we propose a new ML-based dwell time prediction model. In particular, we use a random forest algorithm that significantly outperforms previous heuristics.
- In a comprehensive simulation study, we thoroughly compare our task allocation algorithm and the dwell time prediction to other published heuristics and a theoretical optimum. Our results underline the efficacy of our solution.

The rest of our paper is organized as follows. We review the state-of-the-art related to both task offloading as well as vehicular predictions in Section 2. We present the system model, cost function, and utility function in Section 3. In Section 4, we introduce our proposed task migration algorithm. The ML-based dwell time prediction model is presented in Section 4.4. We discuss selected results from an extensive performance evaluation in Section 5. Finally, we present concluding remarks in Section 6.

2. Related Work

In the following, we review related work for our main contributions: task offloading and dwell time prediction.

2.1. Task Offloading

Previous work on task offloading has focused on various optimization strategies and optimization goals, including reducing energy consumption [12], minimizing latency [13], and monetary cost [14], and increasing task success rates [15]. In the following, we categorize these works based on their own optimization approaches: the combination of task offloading and resource allocation, learning-based, game-theoretic, or mobility-aware task offloading.

2.1.1. Task Offloading and Resource Allocation

Mao et al. [15] proposed a joint optimization strategy considering the current system state, including transmitting energy and CPU resources, to reduce execution costs and task failures. Hossain et al. [13] introduced small-cell MEC to enhance offloading decisions and reduce task execution latency. Tran and Pompili [12] optimized a weighted sum of task completion time and energy consumption using convex and quasi-convex optimization techniques, while Zhao et al. [16] applied a distributed task-offloading algorithm for vehicles in MEC to minimize average delay. Although these approaches demonstrated improved resource efficiency, they primarily used simplified network environments and did not account for complex urban traffic dynamics, which limits their applicability in more realistic settings.

2.1.2. Learning-Based Approaches

Wu et al. [17] developed a hybrid task offloading scheme for vehicles using deep Q-learning (DQN) to minimize both task delay and energy consumption. Qin et al. [18] introduced a multi-vehicle task offloading algorithm using the multi-armed bandit (MAB) theory, which significantly reduced task delay and improved resource utilization. While these learning-based approaches showed considerable performance gains, they were validated in controlled environments that lacked the complexity of real urban traffic. Moreover, the lack of realistic mobility data in the training process may lead to significant deviations in performance when applied in practical scenarios.

2.1.3. Game-Theoretic Approaches

Zhao et al. [19] utilized a Nash equilibrium approach for optimal computation offloading and resource allocation, effectively reducing task processing delays and computation costs. Cheng et al. [14] proposed a pricing mechanism for resource allocation that minimized users' total costs while enhancing the monetary utility of the offloading process. These game-theoretic methods provided effective solutions for resource allocation; however, they were based on assumptions that may not hold in more dynamic environments.

2.1.4. Mobility-Aware Approaches

Cha et al. [20] designed a virtual edge formation framework to predict link duration between vehicles and reduce offloading failures due to unexpected disconnections. Zou et al. [21] proposed a collaborative task offloading strategy for vehicles using MEC, effectively reducing energy consumption by leveraging nearby edge resources. However, both approaches were validated in simplified scenarios, such as straight roads or unrealistic highways, which do not fully capture the complexity of urban traffic conditions.

Despite these advancements, challenges remain in task offloading, particularly in dynamic and unpredictable vehicular environments. In this regard, we advance the state of the art by integrating a more sophisticated mobility prediction model exploiting real urban data, to further improve task offloading efficiency, success, and applicability in real-world settings.

2.2. Dwell Time and Trajectory Prediction

In dynamic vehicular environments, mobility prediction models can enhance the offloading process by forecasting relevant metrics such as vehicle trajectories and dwell time. Targeting dwell time estimation, Pannu et al. [22], Schettler et al. [23] utilized artificial neural network (ANN), random forest, and an empirical approach to predict vehicles' dwell time in the Luxembourg city [10] and in Manhattan grid scenarios. However, these models focus solely on dwell time prediction without integrating them into the task offloading or migration strategies.

Other offloading approaches focus on predicting vehicle trajectories [24], location forecasts [25], or overall traffic flow [26]. Xu et al. [26] propose a graph-weighted convolution network for traffic flow prediction. However, this approach diverges from our needs on migration algorithms, where precise dwell time estimation for individual vehicles in stationary VMCs, where the VMC region is fixed and pre-defined), is crucial. General traffic flow statistics do not provide the level of granularity we require for our dwell prediction model for each individual vehicle.

Guo et al. [24] develop a dual LSTM-based model to predict vehicle trajectories for improving task offloading in software-defined vehicular cooperative networks. However, the LSTM-based model is complex and requires significant computational resources for both training and inference. In contrast, our random forest model is simpler, requires less computational power, and is thus more practical for real-time deployment. Zhang et al. [25] use an extended Kalman filter to predict vehicle locations, relying on linear approximations that were not validated in a real-world urban traffic setting. Their experiments involved small unmanned vehicles operating indoors, which fails to capture the complex, non-linear mobility patterns typical of urban environments. In contrast, our dwell time prediction model is trained and predicts the realistic dwell time of vehicles effectively. Lv et al. [27] propose a DQN-based task offloading scheme in the handover area of vehicular edge computing networks based on trajectory prediction. Similarly, the complexity of a DQN model makes real-time decision-making challenging in dynamic vehicular environments, and the lack of real-world validation limits its robustness.

2.3. Open Research Gap and Contributions

Most current research in vehicular task offloading neglects accurate dwell time prediction, which we will show in this paper is essential to ensure task completion without failure. Also, task failure rates are often overlooked as an evaluation metric, which is critical in vehicular environments, where tasks often have hard deadlines. To address these gaps, this work introduces a novel heuristic for task offloading approach aimed at minimizing task completion time and reducing task failures. We incorporate vehicle dwell time prediction using a random forest model, ensuring tasks are offloaded only when vehicles will remain within the VMC for the entire task duration. Our utility function optimizes both task completion time and task failure rates, making the solution closer to real-world requirements.

Acronym	**Description**
ANN	artificial neural network
CPU	central processing unit
C-V2X	cellular V2X
DQN	deep Q-learning
EDF	earliest deadline first
FCFS	first-come first-served
ITS	intelligent transportation system
LSTM	long short-term memory
MAB	multi-armed bandit
MAE	mean absolute error
MB	megabyte
MEC	mobile edge computing
MI	million instructions
MIPS	million instructions per second
ML	machine learning
MSE	mean squared error
RF	random forest
SUMO	simulation of urban mobility
V2X	vehicle-to-everything
VMC	vehicular micro cloud

Table 1: List of relevant acronyms

3. System Model, Cost Function, and Utility

3.1. System Model

The system incorporates four elements: VMC, vehicles, tasks, and a controller. The VMC comprises dynamic vehicles in a certain region. Vehicles can act as both task generators and mobile edge servers with varying computational capabilities, enabling efficient task processing. Tasks generated by vehicles are heterogeneous in data size, computational complexity, and deadlines. The central controller predicts vehicular dwell times and prepares task migrations schedules. An example is illustrated in Figure 1. There are five cars in this VMC and the tasks generated by cars have identical colors with the generator. Car A processes locally three generated tasks and migrates two tasks to Car B for remote processing. Similarly, Car C executes two local tasks and offloads three tasks to Cars D and E, respectively. Additionally, Car D migrates one task to Car C. Our task migration strategy re-arranges computing resource allocation and changes the task completion delay.

3.1.1. Vehicular Micro Cloud (VMC)

A VMC represents a collection of cars and their computational resources that are in close physical proximity [6, 7]. All cars in a VMC can have multiple roles: They can act as mobile edge servers with diverse computational abilities, and as users requiring reliable and efficient task-processing services.

Variable	**Interpretation**
a	the unit time interval
$Dwell(i, t)$	the estimated dwell time of $\mathsf{Veh}(i)$ at time t
$\mathcal{D}$	datasets of vehicular information
$\mathcal{D}_{train}$	training sets for ML
$\mathcal{D}_{test}$	testing sets for ML
d_{max}	Maximum tree depth for ML
δ	maximum local task completion time in the past
f_{max}	Maximum features for each split for ML
F	the constructed feature set of vehicles
$g(k)$	the generator vehicle of task k
i	the identification number of a car
j^*	the vehicle having the least workload in the set $\mathsf{OPT}(k)$
k	the identification number of a task
L	list of all newly generated tasks
L'	sorted list of all newly generated tasks
$MI(k)$	the complexity of task k
$MIPS(i)$	the computing capacity of car i
N_{trees}	Number of trees for ML
$\mathsf{OPT}(k)$	the set of processor vehicles with the minimum completion time for $\mathsf{Tsk}(k)$
$p(k)$	the processing vehicle of task k
$r(k)$	the result receiving vehicle of task k
R	transmission data rate
$S(k)$	the data size of task k
s_{split}	Minimum samples required to split for ML
s_{leaf}	Minimum samples required at leaf node for ML
T_k	the decision tree in random forest learning for ML
$\mathsf{Tsk}(k)$	task k
t	the current time step
t_i^{enter}	the timestep when car i enters the VMC
t_i^{exit}	the timestep when car i exits the VMC
$t_{gen}(k)$	the generation time of task k
$t_{ddl}(k)$	the completion deadline of task k
$T^{\text{c}}(k, p(k))$	the migration time of task k from $g(k)$ to $p(k)$
$T^{\text{p}}(k, p(k))$	the processing time of task k
$T^{\text{w}}(k, t, p(k))$	the waiting time of task k to be processed in car $p(k)$ at time t
$T(k, j)$	the completion time for $\mathsf{Tsk}(k)$ executed by $\mathsf{Veh}(j)$
$\mathsf{Veh}(i)$	car i
Veh_t	set of vehicles in the current VMC
(x, y)	coordinates of vehicles
$(x, y)_t$	coordinates of vehicles at time t
X	the feature set of vehicles
$\hat{Y}$	the final prediction of all dwell times
Y	the accurate dwell time set

Table 2: Used variables

Three main characteristics of tasks generated by vehicles within the VMC significantly impact our migration performance: task data size, execution complexity, and associated deadline. The task data size impacts the uplink and downlink delays between the task generator (user) and its remote processor (server); execution complexity influences the actual processing time; and the deadline indicates the maximum available computation and communication time.

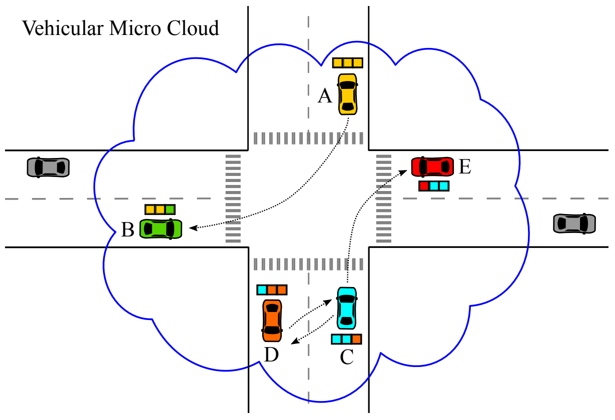


Figure 1: Illustration of the VMC. Vehicles that participate in the VMC are color coded, with corresponding tasks and schedules. Each vehicle can act both as user and server. Vehicle A is processing most of its tasks locally and has two tasks offloaded to vehicle B. Vehicle C has offloaded its tasks to D and E, and it is processing a task of D.

3.1.2. Vehicle Model

The presence of vehicles, their computing capabilities, and their current resource schedule in a VMC fluctuate over time, requiring migration decisions to be made at each time step to ensure optimal task assignment. We use t and i to denote time steps and individual cars, respectively. Thus, the tuple $(t_i^{\text{enter}}, t_i^{\text{exit}})$ indicates the time a car is available in the VMC. The total number of vehicles that ever entered the VMC at any given time is represented by $\max\{i\}$. As for the tasks generated by each vehicle at each time step, we assume this number follows a uniform distribution in $[0, n]$, and a central controller (per VMC) is assumed to receive all updates at each time step. Vehicles are assigned varying computing capabilities, measured in million instructions per second (MIPS). Some studies define computing power in terms of CPU cycles, but using MIPS provides a more straightforward measure since one cycle can execute multiple instructions.

3.1.3. Task Model

We use k to denote the sequence number of a task. Thus, $\max\{k\}$ represents the total number of tasks generated by vehicles that have been part of the VMC. Tasks are heterogeneous in terms of data size, execution complexity, and deadlines. To ensure compatibility with the vehicles' computing capacities, we measure task complexity in million instructions (MI). Additionally, task data sizes indicate the amount of data that must be transmitted for task execution, measured in megabytes (MB). We assume task deadlines are uniformly distributed. The overall delay T_k of task k comprises communication T_k^{c}, waiting for execution T_k^{w}, and processing T_k^{p} where $T_k \triangleq T_k^{\text{w}} + T_k^{\text{c}} + T_k^{\text{p}}$. For simplicity, we assume that no congestion or collision occurs in communication.

3.1.4. Controller Model

The central (per VMC) controller is responsible for collecting information, listing tasks according to their deadlines, calculating potential task completion times, training vehicular dwell time prediction models, predicting dwell times, and making task migration decisions. Without loss of generality, we assume such a controller always exists. This role can be carried out by one of the cars in the VMC, and handed over if the car leaves.

Information collection and dwell time prediction involves the following steps: (1) For each time slot t, the central controller adds newly joined vehicles (including current location, speed, type, lane, task queue, and computing capacity) to the vehicle set. (2) When a vehicle has left the VMC during the previous time slot $t-1$, the controller removes it from the set. Furthermore, its dwell time is recorded and added to the dataset used to train the dwell time prediction model. (3) For all vehicles in the VMC, the dwell time is predicted to be used in migration decisions. (4) When a vehicle generates new tasks, the central controller collects task information, including data size, computational complexity, and deadlines.

3.2. Preliminaries

Let $\mathsf{VMC} = \{\mathsf{Veh}, \mathsf{Tsk}\}$ be a vehicular micro cloud with Veh being the set of all vehicles and Tsk being the set of all tasks. $\mathsf{Veh}(i)$ represents the i-th vehicle in Veh and $\mathsf{Tsk}(k)$ is the k-th task in Tsk.

For each $\mathsf{Veh}(i)$, we denote the VMC entering time $t_{\text{enter}(i)}$, VMC exiting time $t_{\text{exit}}(i)$, and its computing capability $\mathsf{MIPS}(i)$. Thus,

$$\mathsf{Veh}(i) = (t_{\text{enter}(i)}, t_{\text{exit}}(i), \mathsf{MIPS}(i)). \tag{1}$$

For each $\mathsf{Tsk}(k)$, we denote its generation time $t_{gen}(k)$, computing complexity $\mathsf{MI}(k)$, data size $S(k)$, its generator vehicle index $g(k)$, its processing vehicle index $p(k)$, completion deadline $t_{ddl}(k)$, and its result receiver vehicle $r(k)$.[1] That is

$$\mathsf{Tsk}(k) = (t_{\text{gen}}(k), \mathsf{MI}(k), S(k), g(k), p(k), t_{ddl}(k), r(k)). \tag{2}$$

where $p(k)$ and $r(k)$ can only be known when the task migration has been decided.

Let $T^{\text{c}}(k, p(k))$ be the uplink time of $\mathsf{Tsk}(k)$ migrating from $\mathsf{Veh}(g(k))$ to $\mathsf{Veh}(p(k))$ and the channel data rate be R. Then for any $\mathsf{Tsk}(k)$, we have

$$T^{\text{c}}(k, p(k)) = \begin{cases} 0, & \text{if } g(k) = p(k) \\ \frac{S(k)}{R}, & \text{otherwise.} \end{cases} \tag{3}$$

In our communication model, as long as $g(k) \neq p(k)$, the communication time remains constant.

[1]To clarity, $r(k)$ represents the car to which the result of the generated task is returned. This can either be the original task-generating car, indicating a successful result return, or remain empty, signifying a failed return.

Let $T^{\text{p}}(k, p(k))$ be the task processing time cost $\mathsf{Tsk}(k)$ processed by $\mathsf{Veh}(p(k))$,

$$T^{\text{p}}(k, p(k)) = \frac{\mathsf{MI}(k)}{\mathsf{MIPS}(p(k)))} \tag{4}$$

Let $T^{\text{w}}(k, t, p(k))$ be the duration of $\mathsf{Tsk}(k)$ waiting for its execution by $\mathsf{Veh}(p(k))$ at time t. The waiting task queue in $\mathsf{Veh}(p(k))$ is $\mathsf{Tsk}(k_0), \mathsf{Tsk}(k_1), \cdots, \mathsf{Tsk}(k_q)$, where $\mathsf{Tsk}(k_0)$ is under processing and the other tasks are waiting for execution. Then

$$T^{\text{w}}(k, t, p(k)) = \frac{(1 - \alpha_{p(k),t})\mathsf{MI}(k_0)}{\mathsf{MIPS}(p(k))} + \sum_{n=1}^{q-1} T^{\text{p}}(k, p(k)), \tag{5}$$

where $\alpha_{i,t}$ implies the proportion of instructions that have been finished of $\mathsf{Tsk}(k_0)$.

3.3. Cost Function and Utility Function

Our optimization problem can be formulated in a simplified form by minimizing a time cost function without accounting for task failure, or more comprehensively by maximizing a utility function. The time cost function aims to minimize the total task completion time. The utility function is designed to balance multiple factors by combining the average task completion time with two weighted metrics for task failure: the proportion of tasks that miss their deadlines and the proportion of tasks where the generating or processing device exits the VMC before task completion.

We assume a $\mathsf{Tsk}(k)$ is generated and migrated at time t to $\mathsf{Veh}(p(k))$, where there have been already $q-1$ tasks in the waiting queue and $\mathsf{Tsk}(k)$ is the q^{th} task in the queue.

The expression $T^{\text{sum}}(k, t, p(k))$ represents the completion time of task $\mathsf{Tsk}(k)$ as

$$\begin{aligned} T^{\text{sum}}(k, t, p(k)) =& T^{\text{c}}(k, p(k)) + T^{\text{p}}(k, p(k)) + \\ & T^{\text{w}}(k, t, p(k)) \\ =& \mathbb{I}(p(k) \neq g(k)) \frac{S(k)}{R} + \frac{\mathsf{MI}(k)}{\mathsf{MIPS}(p(k))} + \\ & \frac{(1 - \alpha_{p(k),t})\mathsf{MI}(k_0)}{\mathsf{MIPS}(p(k))} + \sum_{n=1}^{q-1} T^{\text{p}}(k_n) \end{aligned}$$

It consists of communication time $T^{\text{c}}(k, p(k))$, processing time $T^{\text{p}}(k, p(k))$, and the waiting time $T^{\text{w}}(k, t, p(k))$. The communication cost is positive only if the task is not executed locally, i.e., $p(k) \neq g(k)$. Therefore, we have

$$\begin{aligned} T^{\text{sum}}(k, t, p(k)) =& \mathbb{I}(p(k) \neq g(k)) \frac{S(k)}{R} + \\ & \sum_{n=1}^{q} \frac{\mathsf{MI}(k_n)}{\mathsf{MIPS}(p(k))} + \frac{(1 - \alpha_{p(k),t})\mathsf{MI}(k_0)}{\mathsf{MIPS}(p(k))} \end{aligned} \tag{6}$$

The optimization problem can now be formulated as

$$\begin{aligned} \underset{k}{\text{minimize}} \quad & T^{\text{sum}}(k, t, p(k)) \\ \text{subject to} \quad & t \geq 0, \quad k, p(k), q \geq 1 \end{aligned} \tag{7}$$

The sum of completion time for N tasks over a certain duration δ is

$$T^{\text{sum}}(N, \delta) = \sum_{k=1,t=0}^{k=N,t=\delta} (k, t, p(k)) \tag{8}$$

Before introducing the utility function, the failure rate (caused by unpredicted leaving vehicles) $F_m(N, \delta)$, and the failure rate (caused by missing task deadlines) $F_d(N, \delta)$ over a certain time interval δ with N tasks generated in VMC are respectively defined as:

$$\begin{aligned} F_m(N, \delta) &= \frac{\sum_{k=1}^{N} \mathbb{I}(r(k) \neq g(k))}{N} \\ F_d(N, \delta) &= \\ & \frac{1}{N} \sum_{k=1}^{N} \mathbb{I}(T^{\text{sum}}(k, t, p(k)) > (t_{ddl}(k) - t_{gen}(k))) \end{aligned} \tag{9}$$

Moreover, these two task failure rates are independent. For instance, if a task is completed before its deadline but its generator has left before it is completed, then the task failure can only be accounted into F_m, but not into F_d.

The overall utility function can now be formulated as

$$\begin{aligned} U(N, \delta) = & -\frac{1}{N} T^{\text{sum}}(N, \delta) - \beta \frac{\sum_{k=1}^{N} \mathbb{I}(r(k) \neq g(k))}{N} - \\ & \sigma \frac{\sum_{k=1}^{N} \mathbb{I}(T^{\text{sum}}(k, t, p(k)) > (t_{ddl}(k) - t_{gen}(k))}{N} \end{aligned} \tag{10}$$

where β and σ are the penalty coefficients of tasks being unsuccessfully returned or missing deadlines, respectively.

By minimizing the cost function, we determine the least completion time for a single task. Our approach uses a greedy heuristic that evaluates all potential processors, including the task generator, to offload the task to the least time-consuming option. This procedure is repeated for each subsequent task to achieve the shortest offloading time. The primary difference between the cost function and the utility function lies in their scope: the cost function focuses on minimizing the completion time for a single task, forming the foundation for the system utility function. In contrast, the utility function considers both delay and failure costs for all tasks within the VMC over a specified duration. Locally processed tasks are guaranteed to be completed without failure caused by unpredictably leaving vehicles, though they may still miss deadlines.

4. Comprehensive Task Migration Algorithms

We designed a task migration algorithm taking the following objectives into consideration. First, we aim to realize the sequence of tasks being selected for offloading based on their deadlines (Algorithm 1). Second, the central controller steers the offloading process, taking dynamic vehicular dwell time prediction into account (Algorithm 2). Last and essentially, we identify the best vehicle for task migration based on a pre-sorted sequence (Algorithm 3), especially by leveraging future dwell times to reduce the probability of task failures (Algorithm 4).

4.1. Task Pre-Scheduling

The task pre-scheduling function is applied only to tasks generated in the last time step. It prioritizes urgent tasks based on their deadlines, similar to the earliest deadline first (EDF) approach, but excludes tasks generated from previous time steps.[2] By reordering tasks, those with more urgent deadlines are prioritized over those with less strict deadlines. Algorithm 1 depicts all the task pre-scheduling steps.

4.2. Central Task Scheduling

The central task scheduling (Algorithm 2) is designed to optimize task assignment decisions in a VMC by predicting the remaining dwell time of vehicles and leverages the prediction result to dynamically offload tasks to the most appropriate vehicles for remote execution.

4.3. Task Migration

The task migration algorithm concretely manages task offloading based on an ordered list of undecided tasks (Algorithm 3). The controller selects suitable processing cars based on the shortest task completion time within their dwell time constraints and with the least workload. If the original vehicle offers the shortest completion time, the task is assigned for local task processing.

4.4. ML-based Dwell Time Prediction

Accurately predicting the entire dwell time of vehicles from entry to exit in VMC has been shown challenging in [22, 23]. We propose a random forest regression model (cf. Algorithm 4) to estimate only the future remaining time of cars, using previous location information and an adaptive

[2] Please note that we do not support preemption and re-migration at the moment.

Algorithm 1 Task pre-scheduling

Input: L: List of all newly generated tasks
Output: L': Sorted list of tasks by deadline

```
1: function PRE-SCHEDULE-TASKS($L$)
2:     $L \leftarrow$ collect tasks generated by vehicles in VMC at time $t$
3:     $L' \leftarrow$ sort $L$ in increasing order of deadlines
4:     return $L'$
5: end function
```

Algorithm 2 Central scheduling

```
Input: Set of vehicles $Veh_t$ in the current VMC
Output: Predicted dwell times and task migration decisions
1: for each time step $t = 1, \dots, T$ do
2:     for each vehicle $Veh(i)$ in the set of $\mathsf{Veh}_t$ do
3:         Predict dwell time of $Veh(i)$ using Algorithm 4
4:     end for
5:     for each vehicle $Veh(i)$ in the set of $\mathsf{Veh}_t$ do
6:         Execute task migration using Algorithm 3
7:     end for
8: end for
```

Algorithm 3 Task migration

```
Input: Initial vehicle set Veh, task set Tsk
Output: Optimized task assignment decisions
1: for each time slot $t = 1, 2, \dots$ do
2:     for each vehicle $\mathsf{Veh}(i)$ in latest Veh set do
3:         Collect predicted dwell time $Dwell(i)$ of $\mathsf{Veh}(i)$
4:         Collect new and already assigned tasks
5:     end for
6:     Controller forms list of unassigned tasks $L$
7:     $L' \leftarrow$ sorted list of $L$ from Algorithm 1
8:     for each task $\mathsf{Tsk}(k)$ in $L'$ do
9:         for each vehicle $\mathsf{Veh}(j)$ in Veh do
10:            if $T(k,j) \le Dwell(i)$ then
11:                $T(k,j) \leftarrow T^{c}(k,j) + T^{p}(k,j) + T^{w}(k,t,j)$
12:            end if
13:        end for
14:        $\mathsf{OPT}(k) \leftarrow \mathrm{argmin}_j\, T(k,j)$
               ▷ Find vehicles with minimum completion time for $\mathsf{Tsk}(k)$
15:        if $g(k) \in \mathsf{OPT}(k)$ then    ▷ $g(k)$ is preferred vehicle for $\mathsf{Tsk}(k)$
16:            $p(k) \leftarrow g(k)$
17:        else
18:            Find $j^* \in \mathsf{OPT}(k)$ with minimum workload
               ▷ Select vehicle with least load
19:            $p(k) \leftarrow j^*$
20:        end if
21:    end for
22: end for
```

upper bound. The model predicts the future dwell time that will suffice for task completion, disregarding any exceeding time longer than needed. By collecting the maximum task completion time denoted as δ over a time window $[t-\Delta, t]$, the upper bound δ serves as an adaptive threshold. Used features include the current time t, current position (x, y), and past positions sampled at time steps up to δ, ensuring sufficient temporal context for accurate predictions.

Our proposed random forest model predicts the future dwell time of each vehicle at each time step. Initially, it prepares a set of features, including the current time and its previous and current location, the previous time has

Algorithm 4 Random forest dwell time prediction

```
1: Input: Floating car data collected over a certain time duration $T$
2: Output: Predicted future dwell time $Dwell(i)$ for each vehicle at each time step
3: $\delta \leftarrow$ maximum local task completion time in the past
4: $a \leftarrow$ interval between two time steps    ▷ we use $a = 0.2\,\mathrm{s}$ in this paper
5: $t \leftarrow \{t-T, t-T+a, t-T+2a \dots, t-2a, t-a, t\}$, where $t$ is the current time step
6: for each vehicle $Veh(i)$ do
7:     for each time step $t$ in the duration of $[t-\delta+a, t]$ do
8:         Extract coordinates $(x, y)$ for $t-a$
9:     end for
10:    Calculate the dwell time of $Veh(i)$ at $t$ and upper bound it by $\delta$
11: end for
12: Feature preparation:
13: Construct feature set $F = \{t, (x,y)_t, (x,y)_{t-a}, \dots, (x,y)_{t-\delta}\}$
14: Split dataset $\mathcal{D}$ into training $\mathcal{D}_{train}$ and testing $\mathcal{D}_{test}$ sets
15: Model initialization and hyperparameter tuning:
16: Define random forest hyperparameters ($N_{trees}$, $d_{max}$, $s_{split}$, $s_{leaf}$, $f_{max}$) to optimize with Optuna
17: Training process:
18: for $k = 1, \dots, N_{trees}$ do
19:     Train decision tree $T_k$ on the training set $\mathcal{D}^{(k)}_{train}$
20: end for
21: The final prediction $\hat{Y}$ is
        $$\hat{Y} = \frac{1}{N_{trees}} \sum_{k=1}^{N_{trees}} T_k(F)$$
22: The predicted future dwell time of $Veh(i)$ is
        $$Dwell(i) = \hat{Y}_i$$
23: Model evaluation:
24: Compute MSE and MAE on $\mathcal{D}_{test}$:
        $$\mathrm{MSE} = \frac{1}{|\mathcal{D}_{test}|} \sum_{(X_i,Y_i)} \in \mathcal{D}_{test} (Y_i - \hat{Y}_i)^2$$
        $$\mathrm{MAE} = \frac{1}{|\mathcal{D}_{test}|} \sum_{(X_i,Y_i)} \in \mathcal{D}_{test} |Y_i - \hat{Y}_i|$$
25: Model deployment:
26: Save trained model $f_{RF}$ for future predictions
```

threshold δ, and then it splits the dataset into training and testing sets. Key hyperparameters of the random forest model are N_{trees}: number of trees; d_{max}: maximum tree depth; s_{split}: minimum samples required to split; s_{leaf}:

Model	**Scenario**		**MSE** [s^2]	**MAE** [s]
Proposed	Single intersection	Low	0.003	0.015
		Medium	0.006	0.028
		High	0.008	0.037
Proposed	Luxem-bourg	Suburb	0.008	0.046
		Highway	0.021	0.110
		City	0.008	0.046
Proposed	Nagoya	Suburb	0.066	0.027
		State	0.030	0.026
		City	0.013	0.014
Johnson SU	Single intersection	Low	552.727	18.976
		Medium	491.971	17.680
		High	648.250	18.552
Johnson SU	Luxem-bourg	Suburb	0.615	0.633
		Highway	1177.998	27.880
		City	790.667	17.157
Johnson SU	Nagoya	Suburb	953.707	28.740
		State	690.588	21.335
		City	367.292	15.387

Table 3: Performance metrics of dwell time prediction across different models and scenarios (time step 0.2 s).

minimum samples required at leaf node; f_{max}: maximum features for each split. We used the Optuna hyperparameter optimization framework[3] for training and optimization.

During training, the algorithm creates multiple decision trees using bootstrapped samples. Each tree in the forest maps the input features (time and locations from δ seconds ago till now) to the future dwell time. The random forest model then makes predictions $\hat{y}$ by averaging the future dwell time outputs from all decision trees. The algorithm evaluates the model performance by mean squared error (MSE) and mean absolute error (MAE). Finally, the trained model is saved for future dwell time predictions.

We validate our random forest model for dwell time estimation using three scenarios (cf. Section 5.1). Using the Optuna hyperparameter optimization framework, our proposed model achieves a MSE of less than 0.06 s^2 and a MAE of less than 0.03 s. In comparison, the Johnson SU distribution performs less effectively with a MSE of up to 1177 s^2 and a MAE of up to 29 s, which is in line with the results reported in [23]. The performance comparison is presented in detail in Table 3.

4.5. Example of Operation and Underlying Conflicts

For further clarification, concretely, all newly generated tasks are ordered at each step based on their generation time. For instance, at time step t, tasks generated within the interval $(t-a, t]$ are considered.[4] Let's assume that only two tasks are generated during the interval $(t-a, t]$: task k, generated at $t-a+0.001$ seconds with a deadline of $t+2$ seconds, and task $k+1$, generated at $t-a+0.002$ seconds with a deadline of $t+1$ seconds. Initially, the tasks are sequentially ordered according to their generation times in the list L. However, as described in Algorithm 1, tasks generated during $(t-a, t]$ are re-sorted in the list L' at time t based on their priority, determined by their deadlines. Consequently, task $k+1$ is prioritized for offloading, as it has an earlier deadline.

In the decision-making process, the selected task's $k+1$ completion time is calculated for each potential offloading option. This time contains the communication time (only required for remote offloading), the waiting time for execution on a potential processor vehicles, and the processing delay give the different computing resources. Next, task $k+1$ is offloaded to the vehicle that minimizes its completion time. Meanwhile, the remaining computing resources are updated. Possible completion times of task k are calculated and it is offloaded accordingly. When an underlying conflicts occur, i.e., multiple tasks have the same deadline in the same time slot, the one that has been generated earlier will be prioritized in the list L'.

[3]https://github.com/optuna/optuna

[4]Here, a represents the interval between time steps, which is set to 200 ms in our experiments.

Reference	Data size [MB]	Complexity [MI]	CPU [MIPS]
Chen et al. [28]	0.5	1000	500–10 000
Jang et al. [29]	1.25–3.125	-	-
Mao et al. [15]	0.125	0.7375	1500
Liu et al. [30]	0.1–0.3	100–400	1000–5000
Bute et al. [31]	0.0125–0.125	1000–5000	1000–4000
Shi et al. [32]	0.02–0.2	200–3200	5000–10 000
Our evaluation	0.1–1	1000–5000	1000–5000

Table 4: Typical task properties used in the literature. If a range is given, a uniform distribution is assumed.

5. Simulation Setup and Results

In the following, we first present the simulation setup we used followed by an in-depth look at selected results.

5.1. Simulation Parameters

Selecting representative parameters for task settings remains a challenge. This includes task types, data size, deadlines, and computing resource requirements. Following a literature study, we found that there is a huge variety of applications considered. Table 4 gives an overview of typical settings. If a range is given, a uniform distribution is assumed. Based on this overview, we selected the parameters for our performance evaluation (last row in the table). In addition, we assume task deadlines in the range 1–1.5 s, which conceptually matches our time slots.

Without loss of generality and also following the observations in [9], we set the data rate to 50 Mbit/s, which can be easily supported by most current communication technologies. For accurate mobility simulation, we used SUMO Version 1.19.0[5] [33]. For best comparability, we use three different traffic scenarios.

Parameter	Low	Med	High
Total number of cars	480	900	1200
Average speed [km/h]	27.86	25.74	21.20
Average dwell time [s]	865.16	1002.77	887.46
Simulation duration [s]	3600	3600	3600
Car generation period [s]	7.5	4	3
Tracking Time Step [ms]	200	200	200

Table 5: Road traffic parameters: simple intersection scenario

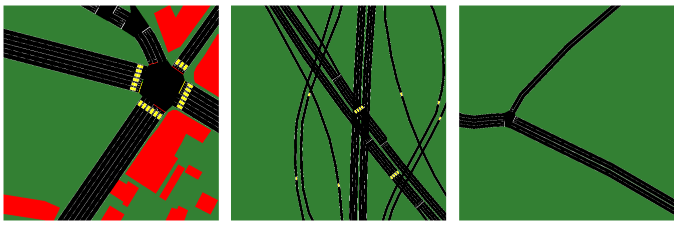

(a) City (b) Highway (c) Suburban

Figure 2: Luxembourg scenario

5.1.1. *Simple intersection*

For ease of validation, we use a single intersection scenario. This scenario contains a single traffic light, four entry points, and 16 possible routes, including straight paths, right turns, left turns, and U-turns. The map covers an area of approximately $400m \times 400m$. Trips were generated using the SUMO random trip generator, which allows for flexible control over car generation periods, thereby affecting car density. After generating mobility for three different traffic densities (low, medium, high), the simulation outputs floating car data, including position, speed, lane, and other information about every vehicle at each time step in the simulation. Important characteristics such as the total number of cars, average speed, and average duration of cars can be analyzed and obtained from the traces output. The key parameters are summarized in Table 5.

5.1.2. *Luxembourg scenario*

The Luxembourg SUMO Traffic (LuST) Scenario [10] is a highly cited, detailed, and realistic traffic simulation model for the mid-sized European city of Luxembourg. This scenario provides an ideal environment for testing and evaluating the efficiency and robustness of our algorithm because of its complexity and representativeness of real-world urban traffic conditions. To assess the effectiveness of our migration mechanism in real traffic, we selected three distinct regions within the map as VMC zones, each characterized by varying vehicle flow and density. These zones are strategically located in the city center, on a highway, and in a suburban area (Figure 2). For the SUMO simulations, we chose three representative time periods throughout the day: midnight, morning peak hour, and non-peak afternoon, each reflecting distinct traffic conditions. The key parameters are summarized in Table 6.

Time	Parameter	City	Highway	Suburban
Midnight 00:20	Total number of cars	5	8	0
	Avg. speed [km/h]	16.778	87.846	0
	Avg. dwell time [s]	28.4	16.375	0
	Max. dwell time [s]	77.000	36.000	0
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200
Morning 08:20	Total number of cars	102	168	10
	Avg. speed [km/h]	8.040	89.920	52.730
	Avg. dwell time [s]	37.921	9.125	10.0
	Max. dwell time [s]	99.000	41.000	20.000
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200
Afternoon 16:20	Total number of cars	68	64	2
	Avg. speed [km/h]	11.826	104.970	43.077
	Avg. dwell time [s]	29.191	8.828	10.5
	Max. dwell time [s]	75.000	40.000	15.000
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200

Table 6: Road traffic parameters: Luxembourg scenario

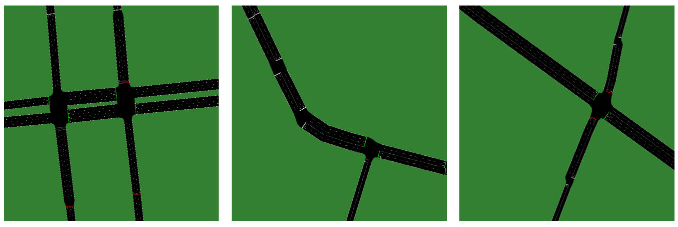

(a) City (b) State road (c) Suburban

Figure 3: Nagoya scenario

Time	Parameter	City	State Road	Suburban
Midnight 00:20	Total number of cars	7	7	3
	Avg. speed [km/h]	20.629	25.659	16.605
	Avg. dwell time [s]	25.571	21.000	22.000
	Max. dwell time [s]	33.000	59.000	43.000
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200
Morning 08:20	Total number of cars	70	48	28
	Avg. speed [km/h]	24.864	22.976	19.090
	Avg. dwell time [s]	17.028	31.520	27.285
	Max. dwell time [s]	58.000	67.000	92.000
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200
Evening 20:20	Total number of cars	21	13	8
	Avg. speed [km/h]	24.802	18.205	17.242
	Avg. dwell time [s]	24.809	41.076	31.000
	Max. dwell time [s]	50.000	66.000	47.000
	Simulation time [s]	100	100	100
	Time Step [ms]	200	200	200

Table 7: Road traffic parameters: Nagoya scenario

[5]https://eclipse.dev/sumo/

5.1.3. Nagoya scenario

We also selected the Nagoya Urban Mobility (NUMo) scenario [11] as one of the most comprehensive urban traffic scenarios, providing a highly realistic vehicle mobility of the entire city of Nagoya, Japan. Featuring a densely populated urban environment with complex traffic patterns, NUMo enhances our vehicular mobility simulations by reflecting a more accurate representation of real-world dynamics compared to other scenarios. NUMo's traffic signals are meticulously calibrated, and its primary data source is actual traffic counts. This makes NUMo an even closer approximation to real-world conditions, making it particularly valuable for our research. Three distinct regions within the map are chosen as VMC zones. These zones are located in the city center, on an entrance of a state road, and in a suburban area (cf. Figure 3). For the SUMO simulations, we again chose three representative times of the day: midnight, morning peak hour, and non-peak afternoon, each reflecting distinct traffic conditions. The key parameters are summarized in Table 7.

5.2. Baselines for Performance Evaluation

5.2.1. No migration

In this approach, tasks are processed locally without any offloading or migration to other vehicles in the same VMC. By comparing task completion times between the *no migration* strategy and migration mechanisms, we can quantify the performance gains achieved through task migration strategies.

5.2.2. Migration based on accurate dwell time

All migration strategies use the same offloading algorithm but rely on different sources for future vehicular dwell times. This baseline assumes accurate knowledge of the dwell time, obtained from the SUMO traces. It is representing the theoretical optimal performance achievable.

5.2.3. Johnson SU distribution of dwell times

The Johnson SU distribution has been empirically identified as a good representation of dwell time distribution in Luxembourg (cf. [22]). Schettler et al. [23] already used it as a baseline in their research. Unlike other prediction approaches, this method does not require training a machine learning model.

5.3. Simple Intersection Scenario

Figure 4 shows the performance of the four task offloading strategies for the simple intersection scenario. We plot the task completion time in Figure 4a and the saved time ratio, i.e., the proportion of time saved by using three migration strategies compared to the no migration strategy, in Figure 4b.

Task completion time, defined as the duration from a task's generation to its completion – whether executed entirely within the VMC or partially outside it – is a straightforward and direct metric. In our setup, each car uniformly randomly generates 0–3 tasks per time-step. For *no migration*, the high-, medium-, and low-density scenarios show a descending order of completion times, which is reasonable. All migration approaches, whether using accurate knowledge or dwell time prediction models, demonstrate a significant decrease in completion delay compared to the *no migration* strategy. The benefit increases with increasing traffic density, i.e., increasing candidates for task migration.

The saved time ratio $\mathrm{T}^{\text{save}}(k,t,p(k))$ is calculated as the proportion of the time difference between *no migration* and migration strategies, divided by *no migration* time as

$$\mathrm{T}^{\text{save}}(k,t,p(k)) = \frac{T^{\text{local}}(k,t,p(k)) - T^{\text{mig}}(k,t,p(k))}{T^{\text{local}}(k,t,p(k))}$$

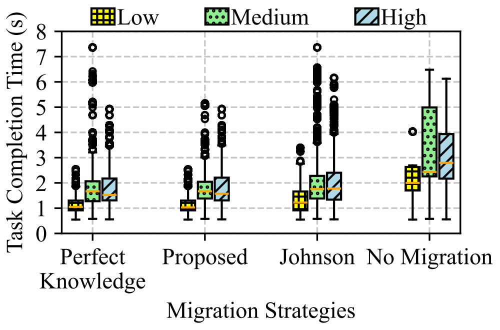
It effectively illustrates the time-saving benefits of task offloading. Regardless of the offloading methods used, the proportions of saved time are similar. Additionally, the medium- and high-density scenarios show a higher proportion of tasks saving around 40 to 50% of the time when migrations using accurate dwell time knowledge or using our random forest model. In contrast, migration decisions based on Johnson SU distribution leads to only below 40% of the saved time ratio.

Figure 4c shows the task failure rate due to missing deadlines, where a task result is completed after its deadline. Figure 4d illustrates the task failure rate due to wrong predictions on cars' mobility, which occurs when the task generator or its processor car leaves the VMC before the task is completed. Both are two independent events. As can be seen, our proposed solution always outperforms Johnson distribution across all scenarios. No migration leads to a high task failure rate due to missing deadlines, and Johnson SU prediction leads to higher failure rates due to wrong mobility prediction.

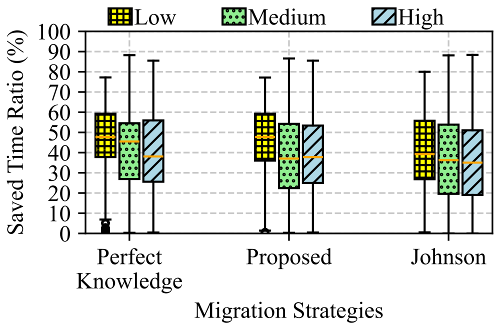
5.4. Luxembourg Scenario

For the Luxembourg scenario, we selected the morning rush hour data for discussion in this paper. During the non-peak period in the afternoon at 4 pm, the algorithm operates similarly to the peak hour. However, during midnight, only very few vehicles are present, limiting task processing to local execution. So, we concentrate on the different locations in the city. The results of our simulations for average task completion time and saved time ratio as well as the task migration failure rate are shown in Figure 5.

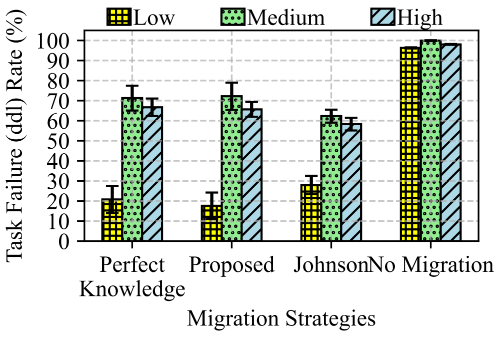
In terms of reducing task completion time, all migration strategies are effective. The results for our proposed algorithm and for the theoretical optimum using perfect dwell time knowledge yield nearly identical performance, demonstrating the superiority of our solution. Again, the Johnson SU based task mitigation results in slightly increased task completion times (Figure 5a) and lower saved time ratios (Figure 5b), but the differences are not significant. However, when evaluating the task failure rate, Johnson SU leads to a pronounced failure rate, whereas our prediction model reduced this by about 36%.



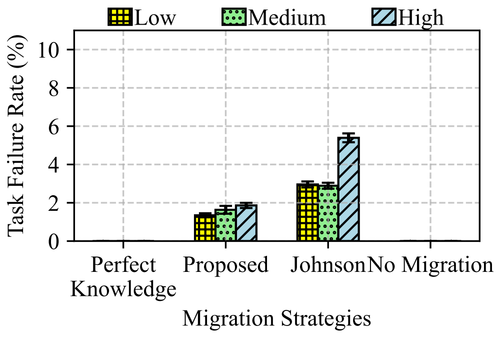
(a) Task completion time



(b) Saved time ratio

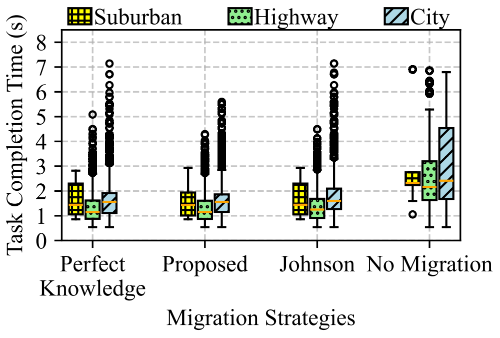


(c) Task failure rate due to missed deadline

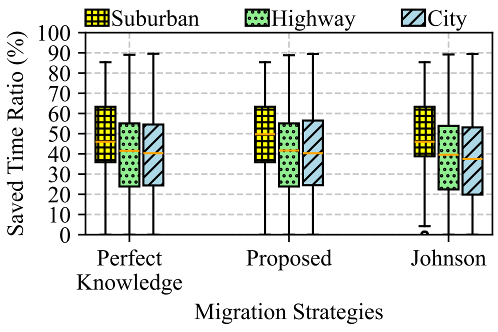


(d) Task failure rate due to mobility prediction

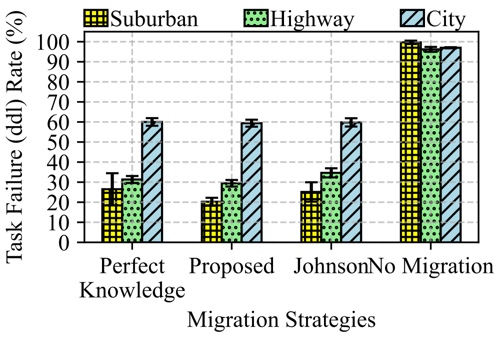
Figure 4: Migration performance, simple intersection scenario



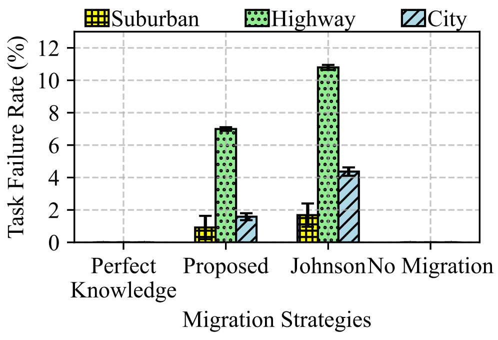
(a) Task Completion Time



(b) Saved Time Ratio

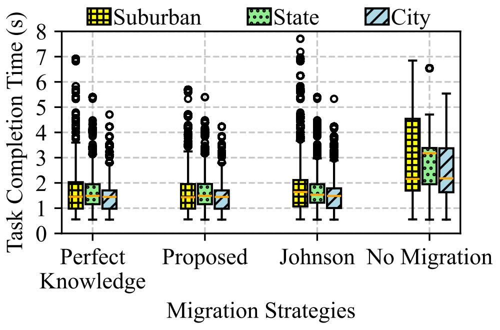


(c) Task failure rate due to missed deadline

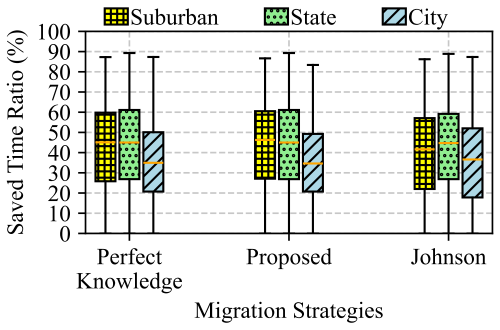


(d) Task failure rate due to mobility prediction

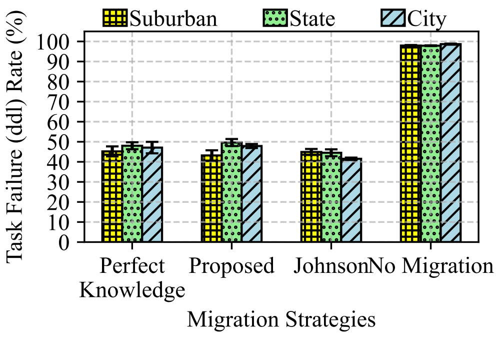
Figure 5: Migration performance, Luxembourg scenario during morning rush hour



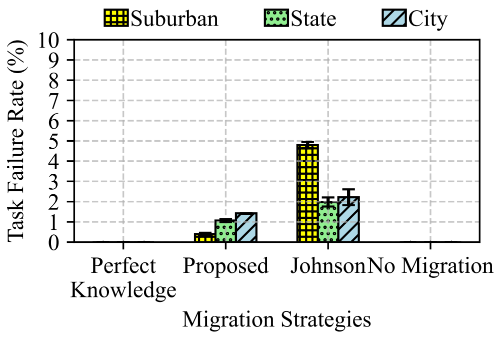
(a) Task Completion Time



(b) Saved Time Ratio



(c) Task failure rate due to missed deadline



(d) Task failure rate due to mobility prediction

Figure 6: Migration performance, Nagoya scenario during morning rush hour

5.5. *Nagoya Scenario*

Also for the Nagoya scenario, we selected the morning rush hour data for discussion in this paper. During the non-peak periods in the evening at 8 pm, the algorithm operates similarly to the peak hour. During midnight, in the city and suburban areas of Nagoya, only one vehicle is present. As a result, the migration strategy requirements are not effective. Thus, again, we focus on the morning rush hour. The results of our simulations for average task completion time and saved time ratio as well as the task migration failure rate are shown in Figure 6.

Task completion times (cf. Figure 6a) across the three migration strategies are significantly reduced compared to no migration (cf. Figure 6b). Our proposed model clearly outperforms the Johnson SU approach w.r.t. the task failure rate (mobility reason, cf. Figure 6d): Our proposed solution exhibits a failure rate of around 0.5–1.5 %, whereas the Johnson SU approach leads to 2.0–5.0 % across all locations in Nagoya. This demonstrates the robustness and effectiveness of our migration mechanism when combined with the dynamic prediction model.

5.6. *Discussion*

An effective task migration strategy in dynamic vehicular environments, involving predicted vehicle dwell times, is crucial for reducing task completion time and minimizing failures. Using a shortest-delay-oriented greedy algorithm significantly reduces delays and deadline misses compared to no migration. Overall, our strategy excels by addressing two key challenges: minimizing task delay and reducing the risk of failure due to vehicle mobility.

Accurate dwell time prediction prevents failures due to unpredictable mobility. The proposed random forest model outperforms previous approaches like Johnson's SU distribution, achieving near-optimal dwell time prediction for offloading. Our random forest model enhances task offloading by selecting candidates that minimize completion time and remain in the VMC until task completion. By combining the shortest-delay-oriented algorithm with close-to-accurate dwell time prediction, our strategy optimizes task offloading, ensuring tasks are assigned to vehicles that minimize completion time and remain in the VMC until completion.

Lessons learned from the evaluation are twofold. First, unlike previous work relying on static assumptions, we used realistic urban traffic data for dwell time prediction, significantly reducing failures due to mobility. Second, we evaluated both time efficiency and task failure metrics, resulting in a comprehensive utility function that aligns with real-world efficiency and reliability concerns.

6. Conclusion

In this paper, we presented a task migration mechanism for vehicular micro clouds (VMCs), focusing on minimizing task completion time and eliminating task failures by utilizing a dynamic dwell time prediction model. Our approach estimates the upper-bounded remaining duration for vehicles within VMCs at each time step, letting optimal offloading decisions without results loss. The proposed mechanism is based on two main pillars: (1) selecting the optimal target for task migration based on the shortest estimated task completion time, and (2) ensuring that both the sender and target vehicles remain within the VMC till task completion by leveraging precise dwell time predictions.

We evaluated our approach using a simple intersection scenario as well as two real-world urban mobility scenarios, Luxembourg and Nagoya. The use of diverse traffic data from different urban conditions and various times of the day provided a realistic and comprehensive validation environment for our task migration strategy and dwell time predictions. The results demonstrate significant reductions in task completion times and the elimination of two types of task failures: unpredicted vehicle leaves and deadline misses. Our findings validate both the effectiveness of our task migration mechanism and the accuracy of the dwell time prediction model.

For future work, we plan to explore how task splitting and task dependencies can be integrated to further enhance task migration performance. Also, applicability to highway scenarios with high-speed vehicles needs to be explored.

References

[1] Christoph Sommer and Falko Dressler. *Vehicular Networking*. Cambridge University Press, 2014. ISBN 978-1-107-04671-9. doi: 10.1017/CBO9781107110649.

[2] S.A. Abdel Hakeem, A.A. Hady, and Hyungwon Kim. 5G-V2X: standardization, architecture, use cases, network-slicing, and edge-computing. *ACM/Springer Wireless Networks*, pages 6015–6041, July 2020. ISSN 1022-0038. doi: 10.1007/s11276-020-02419-8.

[3] Rafael Molina-Masegosa, Javier Gozalvez, and Miguel Sepulcre. Configuration of the C-V2X Mode 4 Sidelink PC5 Interface for Vehicular Communication. In *14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN 2018)*, Shenyang, China, December 2018. doi: 10.1109/msn.2018.00014.

[4] P. Mach and Z. Becvar. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, March 2017. ISSN 1553-877X. doi: 10.1109/COMST.2017.2682318.

[5] Mustafa Emara, Miltiades C. Filippou, and Dario Sabella. MEC-Assisted End-to-End Latency Evaluations for C-V2X Communications. In *European Conference on Networks and Communications (EuCNC 2018)*, Ljubljana, Slovenia, June 2018. IEEE. doi: 10.1109/eucnc.2018.8442825.

[6] Takamasa Higuchi, Joshua Joy, Falko Dressler, Mario Gerla, and Onur Altintas. On the Feasibility of Vehicular Micro Clouds. In *9th IEEE Vehicular Networking Conference (VNC 2017)*, pages 179–182, Turin, Italy, November 2017. IEEE. ISBN 978-1-5386-0986-6. doi: 10.1109/VNC.2017.8275621.

[7] Falko Dressler, Gurjashan Singh Pannu, Florian Hagenauer, Mario Gerla, Takamasa Higuchi, and Onur Altintas. Virtual Edge Computing Using Vehicular Micro Clouds. In *IEEE International Conference on Computing, Networking and Communications (ICNC 2019)*, Honolulu, HI, February 2019. IEEE. ISBN 978-1-5386-9223-3. doi: 10.1109/ICCNC.2019.8685481.

[8] Falko Dressler, Carla Fabiana Chiasserini, Frank H. P. Fitzek, Holger Karl, Renato Lo Cigno, Antonio Capone, Claudio Ettore Casetti, Francesco Malandrino, Vincenzo Mancuso, Florian Klingler, and Gianluca A. Rizzo. V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing. *IEEE Network*, 36(3):24–31, May 2022. ISSN 1558-156X. doi: 10.1109/MNET.001.2100491.

[9] Ziqi Zhou, Youming Tao, Agon Memedi, Chunghan Lee, Seyhan Ucar, Onur Altintas, and Falko Dressler. Optimizing Task Migration Decisions in Vehicular Edge Computing Environments. In *1st IEEE International Conference on Meta Computing (ICMC 2024)*, Qingdao, China, June 2024. IEEE.

[10] Lara Codeca, Raphaël Frank, and Thomas Engel. Luxembourg SUMO Traffic (LuST) Scenario: 24 Hours of Mobility for Vehicular Networking Research. In *7th IEEE Vehicular Networking Conference (VNC 2015)*, Kyoto, Japan, December 2015. IEEE. ISBN 978-1-4673-9411-6. doi: 10.1109/VNC.2015.7385539.

[11] Takamasa Higuchi, Lei Zhong, and Ryokichi Onishi. NUMo: Nagoya Urban Mobility Scenario for City-Scale V2X Simulations. In *15th IEEE Vehicular Networking Conference (VNC 2024)*, pages 17–24, Kobe, Japan, May 2024. IEEE. doi: 10.1109/VNC61989.2024.10575975.

[12] Tuyen X. Tran and Dario Pompili. Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, January 2019. ISSN 1939-9359. doi: 10.1109/TVT.2018.2881191.

[13] Md Delowar Hossain, Luan N. T. Huynh, Tangina Sultana, Tri D.T. Nguyen, Jae Ho Park, Choong Seon Hong, and Eui-Nam Huh. Collaborative Task Offloading for Overloaded Mobile Edge Computing in Small-Cell Networks. In *34th International Conference on Information Networking (ICOIN 2020)*, pages 717–722, Barcelona, Spain, January 2020. IEEE. doi: 10.1109/ICOIN48656.2020.9016452.

[14] Siyao Cheng, Tian Ren, Hao Zhang, Jiayan Huang, and Jie Liu. A Stackelberg-Game-Based Framework for Edge Pricing and Resource Allocation in Mobile Edge Computing. *IEEE Internet of Things Journal*, 11(11):20514–20530, June 2024. ISSN 2327-4662. doi: 10.1109/JIOT.2024.3372016.

[15] Yuyi Mao, Jun Zhang, and Khaled B. Letaief. Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, December 2016. ISSN 0733-8716. doi: 10.1109/JSAC.2016.2611964.

[16] Haitao Zhao, Qixing Zhu, Yue Chen, and Yinyang Zhu. A Research of Task-Offloading Algorithm for Distributed Vehicles. In *IEEE International Conference on Communications (ICC 2020), Workshops*, pages 1–5, Virtual Conference, June 2020. doi: 10.1109/ICCWorkshops49005.2020.9145331.

[17] Chenhao Wu, Zhongwei Huang, and Yuntao Zou. Delay Constrained Hybrid Task Offloading of Internet of Vehicle: A Deep Reinforcement Learning Method. *IEEE Access*, 10:102778–102788, September 2022. ISSN 2169-3536. doi: 10.1109/ACCESS.2022.3206359.

[18] Hao Qin, Guoping Tan, Siyuan Zhou, and Yong Ren. Adaptive Learning-Based Multi-Vehicle Task Offloading. In *IEEE/CIC International Conference on Communications in China (ICCC 2020)*, pages 1033–1038, Chongqing, China, August 2020. IEEE. ISBN 978-1-7281-7328-3. doi: 10.1109/ICCC49849.2020.9238793.

[19] Junhui Zhao, Qiuping Li, Yi Gong, and Ke Zhang. Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks. *IEEE Transactions on Vehicular Technology*, 68(8):7944–7956, August 2019. ISSN 1939-9359. doi: 10.1109/TVT.2019.2917890.

[20] Narisu Cha, Celimuge Wu, Tsutomu Yoshinaga, Yusheng Ji, and Kok-Lim Alvin Yau. Virtual Edge: Exploring Computation Offloading in Collaborative Vehicular Edge Computing. *IEEE Access*, 9:37739–37751, January 2021. ISSN 2169-3536. doi: 10.1109/access.2021.3063246.

[21] Yufei Zou, Li Lin, and Lei Zhang. A Task Offloading Strategy for Compute-Intensive Scenarios in UAV-Assisted IoV. In *5th IEEE International Conference on Electronic Information and*

Communication Technology (ICEICT 2022), pages 427–431, Hefei, China, August 2022. IEEE. ISBN 978-1-66547-212-8. doi: 10.1109/ICEICT55736.2022.9909200.

[22] Gurjashan Singh Pannu, Seyhan Ucar, Takamasa Higuchi, Onur Altintas, and Falko Dressler. Dwell Time Estimation at Intersections for Improved Vehicular Micro Cloud Operations. *Elsevier Ad Hoc Networks*, 122:102606, November 2021. ISSN 1570-8705. doi: 10.1016/j.adhoc.2021.102606.

[23] Max Schettler, Gurjashan Singh Pannu, Seyhan Ucar, Takamasa Higuchi, Onur Altintas, and Falko Dressler. Learning-based Dwell Time Prediction for Vehicular Micro Clouds. In *18th IEEE International Conference on Mobility, Sensing and Networking (MSN 2022)*, pages 542–549, Guangzhou, China, December 2022. IEEE. doi: 10.1109/MSN57253.2022.00091.

[24] Hui Guo, Lan-lan Rui, and Zhi-peng Gao. V2V Task Offloading Algorithm with LSTM-based Spatiotemporal Trajectory Prediction Model in SVCNs. *IEEE Transactions on Vehicular Technology*, 71(10):11017–11032, October 2022. ISSN 1939-9359. doi: 10.1109/TVT.2022.3185085.

[25] Zhiwei Zhang, Zehan Chen, Yulong Shen, Xuewen Dong, and Ning Xi. A Dynamic Task Offloading Scheme Based on Location Forecasting for Mobile Intelligent Vehicles. *IEEE Transactions on Vehicular Technology*, 73(6):7532–7546, June 2024. ISSN 1939-9359. doi: 10.1109/TVT.2024.3351224.

[26] Xiaolong Xu, Chenyi Yang, Muhammad Bilal, Weimin Li, and Huihui Wang. Computation Offloading for Energy and Delay Trade-Offs With Traffic Flow Prediction in Edge Computing-Enabled IoV. *IEEE Transactions on Intelligent Transportation Systems*, 24(12):15613–15623, December 2023. ISSN 1558-0016. doi: 10.1109/TITS.2022.3221975.

[27] Baiquan Lv, Chao Yang, Xin Chen, Zhihua Yao, and Junjie Yang. Task Offloading and Serving Handover of Vehicular Edge Computing Networks Based on Trajectory Prediction. *IEEE Access*, 9:130793–130804, September 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3112077.

[28] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, October 2016. ISSN 1063-6692. doi: 10.1109/TNET.2015.2487344.

[29] Youngsu Jang, Jinyeop Na, Seongah Jeong, and Joonhyuk Kang. Energy-Efficient Task Offloading for Vehicular Edge Computing: Joint Optimization of Offloading and Bit Allocation. In *91st IEEE Vehicular Technology Conference (VTC 2020-Spring)*, pages 1–5, Virtual Conference, May 2020. IEEE. ISBN 978-1-7281-4053-7. doi: 10.1109/VTC2020-Spring48590.2020.9128785.

[30] Yujiong Liu, Shangguang Wang, Qinglin Zhao, Shiyu Du, Ao Zhou, Xiao Ma, and Fangchun Yang. Dependency-Aware Task Scheduling in Vehicular Edge Computing. *IEEE Internet of Things Journal*, 7(6):4961–4971, June 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2020.2972041.

[31] Muhammad Saleh Bute, Pingzhi Fan, Gang Liu, Fakhar Abbas, and Zhiguo Ding. A Collaborative Task Offloading Scheme in Vehicular Edge Computing. In *93rd IEEE Vehicular Technology Conference (VTC 2021-Spring)*, pages 1–5, Virtual Conference, April 2021. IEEE. doi: 10.1109/VTC2021-Spring51267.2021.9448975.

[32] Jinming Shi, Jun Du, Jingjing Wang, Jian Wang, and Jian Yuan. Priority-Aware Task Offloading in Vehicular Fog Computing Based on Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology*, 69(12):16067–16081, December 2020. ISSN 1939-9359. doi: 10.1109/TVT.2020.3041929.

[33] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker-Walz. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.