Latency-Aware and Proactive Service Placement for Edge Computing

Henda Sfaxi, Imene Lahyani, Sami Yangui, *Member, IEEE*, and Mouna Torjmen

***Abstract*—Smart IoT devices and applications in smart cities exchange important real-time information with their environment. However, a subset of these systems may face limitations in analyzing and processing the required large amounts of data to meet ultra-low-latency criteria. This limitation could be attributed to factors such as constrained CPU and battery resources. Thanks to the 5G and edge computing capabilities, a viable solution involves migrating a subset of these latency-sensitive and computationally intensive tasks to edge nodes and servers. This strategic service placement ensures a safe continuity of the application. In this paper, autonomous cars operating in smart cities, engaging in continuous data exchange with their external environment to meet real-time and latency-sensitive requirements, serve as an illustrative example of smart applications. The car's decision service is strategically placed on edge nodes through a proactive (re)placement approach designed for dynamic and mobile environments. This approach uses a quality of service (QoS) metric prediction degradation module, which leverages Exponential smoothing methods to identify a suitable edge node for hosting the car's decision module, with latency as a key criterion. Multiple configurations for outlier detection techniques are evaluated. A proof-of-concept validates the chosen model by comparing it to the AutoRegressive Integrated Moving Average (ARIMA) and the proposed proactive service (re)placement approach. This approach ensures the continuity of the placed module, suggesting the feasibility of locating non-critical modules on edge nodes.**

***Index Terms*—ARIMA, edge computing, quality of service (QoS), service placement, time-series forecast.**

I. Introduction

ACCORDING to [1], the number of megacities with over 10 million inhabitants is projected to increase from the current 21 to 29 by the year 2025. The interconnected IoT devices are expected to increase by 16% by the end of 2023, resulting in 16 billion devices [2]. As mentioned in [3], the number of IoT connections is forecast to grow from 14.6 billion to 30.2 billion in 2027. Furthermore, the widespread implementation of advanced networks promotes the growth of smart cities and related services, including such sectors as smart health and smart agriculture. Several challenges frequently arise in this context, two of which are outlined below. Firstly, numerous connected devices have limited computational and battery capacities [4], [5]. Secondly, contemporary and future mobile and smart service applications, including but not limited to virtual/augmented reality (VR/AR) and connected cars, are latency-sensitive. These applications involve a substantial number of connected devices and sensors, generate a significant amount of data that needs real-time processing, may require considerable computational resources, and demand an important storage capacity [6].

Despite leveraging its substantial storage and computational capabilities, cloud computing cannot guarantee the low-latency and real-time response requirements these applications often call for, as it is typically deployed within the core network [1]. However, edge computing can ensure the criteria mentioned above, owing to the proximity of its nodes and servers provided at the edge network. Furthermore, studies [6] and [7] propose relying on citizens' connected devices as edge nodes to enhance resource availability within the edge network. Hence, smart and mobile applications may utilize cloud resources, edge resources, or a combination of both, depending on their requirements.

On the other hand, enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low-Latency Communication (URLLC) represent two significant commitments in the era of 5G and 6G networks. These services, among others [8], promise to enhance location awareness [6], [9] and communication among connected devices. Specifically, these services are anticipated to streamline interactions between mobile and IoT-based applications and edge nodes and devices, facilitating the implementation of latency-sensitive use cases.

The applications can offload specific tasks to external nodes to harness the computational and storage capacities of cloud and edge computing, on the one hand, and address the constrained computational capacity and battery life of specific connected devices, on the other. This practice is commonly referred to as service placement. Service placement serves various objectives, such as optimizing energy consumption within the system, achieving faster task execution than on the original devices, and fulfilling other intended benefits, as stated by [10]. Various metrics and parameters have to be considered when addressing the service placement problem (SPP). These include the dynamicity of the network, the mobility support of the edge nodes, and optimization strategies such as determining the Quality of Service (QoS) metric(s) to prioritize, as discussed in [7]. Additionally, it is imperative to

Manuscript received 14 December 2023; revised 22 February 2024; accepted 23 February 2024. Date of publication 11 March 2024; date of current version 21 August 2024. The associate editor coordinating the review of this article and approving it for publication was B. Martini. *(Corresponding author: Henda Sfaxi.)*

Henda Sfaxi, Imene Lahyani, and Mouna Torjmen are with the ReDCAD, ENIS, University of Sfax, Sfax 3038, Tunisia (e-mail: Henda.Sfaxi@enis.tn; Imen.Lahyani@enis.tn; Mouna.Torjmen@enis.tn).

Sami Yangui is with the LAAS-CNRS, Université de Toulouse, INSA Toulouse, 31400 Toulouse, France (e-mail: yangui@laas.fr).

Digital Object Identifier 10.1109/TNSM.2024.3375970

guarantee the uninterrupted functionality of the placed service to make it effective.

Enabling seamless mobility for both end-users and edge nodes while maintaining uninterrupted access to desired services and optimal performance stands out as a significant challenge in edge computing, particularly in the context of SPP. Frequent changes in end-user locations can lead to significant delays or packet loss. In such scenarios, services have to be seamlessly transferred from previous edge nodes to new ones or placed back on the original end-user device, ensuring continuous operation without interruptions. Regarding mobility within SPP, the literature review can be categorized into two primary approaches: reactive and proactive techniques. Reactive techniques for service placement involve the detection of deteriorating QoS, QoE, or mobility conditions among end-users and edge nodes as a precursor to making service placement decisions. In contrast, proactive techniques focus on predicting the degradation of fixed QoS and QoE metrics or forecasting the mobility patterns of end-users and edge nodes to anticipate their movement behavior.

This work focuses on a proactive approach to address the service placement problem in edge computing, considering the network dynamicity and mobility of end-users and edge nodes. It provides two key contributions:

- Contribution 1 ($CONTR1$): A prediction model for a QoS metric,
- Contribution 2 ($CONTR2$): A proactive service placement capability utilizing the prediction model (CONTR1) to detect QoS metric degradation in dynamic and mobile edge computing environments.

The proposed predictive module facilitates the selection of a suitable edge node for service (re)placement based on the predicted latency value as a QoS metric. The module selects the edge node with the lowest predicted latency for service placement while ensuring service continuity. As long as the edge node remains responsive and the predicted latency value remains low (i.e., when $predictedLatencyValue \simeq 0$), the service remains on the edge node. In case a degradation is predicted, the service is relocated to a more appropriate edge node if possible; otherwise, it is used within the car.

To achieve ($CONTR1$), Exponential smoothing models are used alongside a set of techniques for detecting outliers to construct the prediction model for a latency QoS metric degradation prediction. The selection of these techniques is based on specific evaluation criteria. Regarding ($CONTR2$), the chosen model is substantiated through a proof-of-concept, which involves a comparison with AutoRegressive Integrated Moving Average (ARIMA) and validates the proposed proactive service (re)placement approach. This proof-of-concept aims to offload a critical service for demonstration purposes only, showing that the car can operate safely and interact with all other nodes in its neighborhood while ensuring it stays autonomous and keeps operating in a proper and safe way. While offloading critical services for next-generation autonomous vehicles is challenging to implement in real-world scenarios, mainly due to security reasons, the authors believe that it is indeed feasible for non-critical modules such as infotainment and multimedia applications.

The remainder of this paper is structured as follows. Section II introduces the motivating use case and its associated requirements. Section III provides a literature review on SPP in edge computing. Section IV gives insight into the essential background and fundamental concepts related to the time series forecast models. Section V introduces the proposed algorithm for latency forecasting and outlier detection. Section VI discusses the configurations for detecting outliers and their interpretation. Section VII details the developed prototype as a proof-of-concept. Finally, section VIII draws on the main conclusions reached by the paper and outlines directions for future research.

II. Motivating Use Case and Requirements

This section introduces the relevant use case, elucidates the underlying motivation, provides a formal problem statement, and outlines the work's requirements.

A. Autonomous Cars in 5G Networks

Connected autonomous cars are poised to replace conventional vehicles in the near future, potentially reducing traffic congestion and minimizing accidents resulting from human errors during driving. These vehicles can also be designed to accommodate seniors and individuals with reduced mobility. Operating within smart cities, autonomous cars continuously exchange vital information with their external environment, supporting the latency-sensitive embedded applications, particularly the decision-making system, in responding to incoming data. The vast data generated by the cars' embedded sensors may exceed the processing capabilities of the onboard systems, leading to challenges in meeting low-latency requirements [11].

5G and edge computing offer various features, including low-latency communication, location awareness, and substantial computing capacity, which can benefit connected devices [6], [9]. Enhanced Mobile Broadband (eMBB) and Ultra-Reliable Low-Latency Communication (URLLC) are key offerings in the 5G and 6G network era, particularly in smart cities. These services enhance communication between connected devices and streamline interactions between autonomous cars and edge nodes, enabling various latency-sensitive applications.

Given these capabilities, autonomous cars can optimize their energy consumption and leverage the computational resources of edge computing by offloading specific tasks to edge nodes [10], among other advantages.

B. Motivations and Research Issues

Autonomous vehicles, equipped with extensive sensor arrays, generate large volumes of real-time data during their journeys, which may exceed the capabilities of their on-board systems to process promptly under low-latency conditions [11]. To address this challenge, one viable scenario involves offloading specific computational tasks associated with vehicle control to external edge nodes. A relevant study [12] conducted within a smart city framework explored a similar concept. In their proposed architecture, the vehicle's

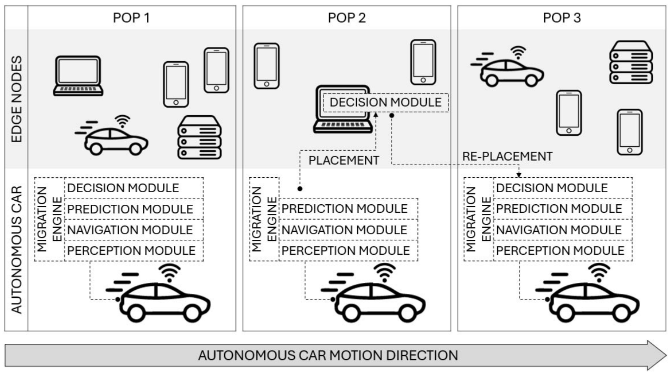


Fig. 1. Autonomous car riding and searching for edge nodes.

operating area was divided into zones known as 'pops.' The autonomous vehicle determined the most suitable edge node for offloading specific computational tasks within each pop. The vehicle's essential modules encompass the decision module (enacting driving policies), the perception module (collecting navigation data), the navigation module (calculating optimal routes), the data collection module (collaborating with other devices to gather real-time information), and the storage module (archiving data generated by other modules). The first four modules are integrated within the autonomous vehicle, while the storage module resides in the 5G core network. Notably, a simulation assessed the viability of migrating the data collection module to various edge nodes within the nearest pop. The results indicated that relocating this module outside the vehicle might not be advantageous.

Given the substantial data volumes the autonomous vehicle generates, which on-board systems may struggle to analyze within low-latency constraints [11], exploring alternative strategies for relocating other modules while ensuring continuous functionality warrants consideration.

C. Problem Formalization

Considering a hypothetical scenario, this study focuses on migrating the autonomous car's decision module to an edge node. As the autonomous car moves from one location (pop) to another, illustrated in Figure 1, it must identify an optimal placement—i.e., an edge node capable of temporarily hosting the latency-sensitive decision module. Some examples of the considered edge nodes in this study include servers, other autonomous cars, and devices owned by citizens.

The decision module plays a vital role in the vehicle's driving engine and must remain consistently available and operational during the car's journey.

Therefore, selecting an edge node with low-latency responsiveness is crucial to ensure the module's continuous availability. If the chosen edge node experiences delays or ceases to respond promptly—possibly due to relocation, low battery, excessive workload, or when the car transitions to a new location—the migrated service should be swiftly relocated to another suitable edge node or back to the car itself.

To address this requirement, a migration engine with a predictive approach first selects an edge node within the current location based on low-latency Quality of Service (QoS) criteria for hosting the decision module. Subsequently, it monitors the QoS metric for potential degradation, preemptively researching and transitioning to another available and suitable edge node. One of the migration engine's primary roles is to ensure the continuous operation of the decision module, which justifies using a predictive technique.

In this study, "latency" refers specifically to network latency, encompassing edge nodes and edge servers collectively referred to as "edge nodes" without differentiation.

D. Requirements

To fulfill the paper's objectives, three distinct requirements are defined:

- Requirement 1 (*REQ*1): The placement service must accommodate the dynamic nature of the network, addressing its evolving topology during runtime.
- Requirement 2 (*REQ*2): The placement strategy should consider the potential mobility of both the computing edge node (where services are hosted and executed) and the end-users (e.g., autonomous vehicles in the use case).
- Requirement 3 (*REQ*3): The placement system should exhibit adaptability and possess the capability to recover from potential degradations in prediction quality.

III. Literature Review

The literature extensively addresses service placement problems in edge computing, proposing various solutions. Ensuring continuous service access and optimal user performance presents a significant challenge, particularly when factoring in end-user and edge node mobility. This section provides an event-based overview and classification of the reviewed literature on SPP. Events stem from the mobility of end-users, edge nodes, and network dynamicity. The classification, summarized in Figure 2, is divided into two main categories: one focusing on end-user and edge node mobility management and the other addressing network dynamicity management. Two primary approaches are considered within the network mobility management category: reactive (triggered by metric degradation) and proactive (predictive metric degradation). The section also compares the paper's contributions and directly related works in the reviewed literature.

A. Edge/User Mobility Management in an Edge Computing Context

Ensuring seamless mobility for both end-users and edge nodes and maintaining uninterrupted access to desired services and optimal performance presents a considerable challenge within edge computing. Frequent relocations of end-users or edge nodes can lead to significant delays or packet loss. Therefore, a robust system must possess the capability to seamlessly (re)place services from the previous nodes to the new ones, ensuring continuous service delivery. In the realm of mobility, the literature review categorizes approaches into two main types: (1) reactive techniques and (2) proactive techniques. Proactive techniques aim to predict the degradation of fixed QoS and QoE metrics and anticipate the mobility

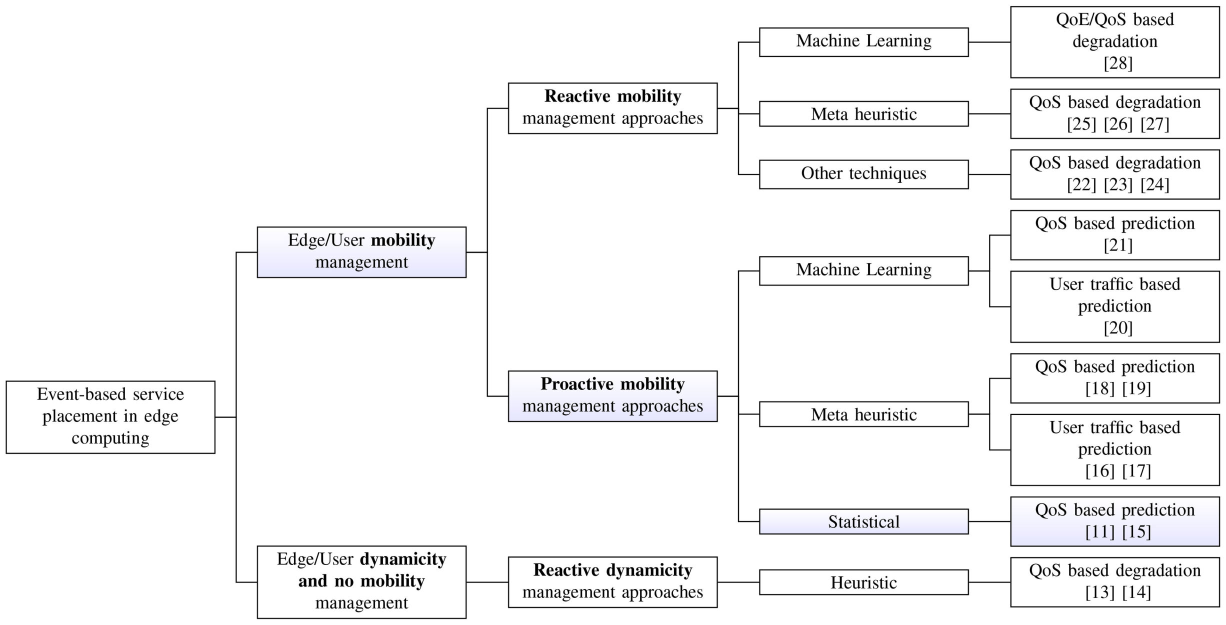

Event-based service placement in edge computing
Edge/User mobility management
Edge/User dynamicity and no mobility management
Reactive mobility management approaches
Proactive mobility management approaches
Reactive dynamicity management approaches
Machine Learning
Meta heuristic
Other techniques
Machine Learning
Meta heuristic
Statistical
Heuristic
QoE/QoS based degradation [28]
QoS based degradation [25] [26] [27]
QoS based degradation [22] [23] [24]
QoS based prediction [21]
User traffic based prediction [20]
QoS based prediction [18] [19]
User traffic based prediction [16] [17]
QoS based prediction [11] [15]
QoS based degradation [13] [14]

Fig. 2. Literature works classification.

patterns of end-users and edge nodes to understand their movement behavior. Existing service management methods are predominantly reactive, where service placement decisions are made in response to detected QoS and QoE degradation or mobility issues concerning end-users and edge nodes.

This review covers relevant works in both reactive and proactive approaches within the context of two edge computing implementations [29], Fog Computing (FC) and Multi-access Edge Computing (MEC). These implementations differ notably, with FC emphasizing medium context awareness, while MEC leverages high context awareness due to the proximity of devices functioning as intermediate edge nodes.

1) Reactive Mobility Management Approaches: Various works aspire to tackle the issue of end-user mobility by offering dynamic[1] placement strategies. The study of [7] classifies different approaches of dynamic placement services in the fog context into supporting mobility or not. Regarding the techniques used to ensure service placement with a reactive approach, the works can be classified into three categories.

a. *Works using Machine Learning for SPP:* Authors in [28] propose an Edge Cognitive Computing (ECC) architecture that ensures ultra-low latency and high user experience. They use a Q-learning method to find the optimal node to host the service.

b. *Works using Metaheuristic for SPP:* Other works, such as [27] and [26], solve the SPP in an online learning mode using the Multi-armed Bandit (MAB) theory. However, they did not consider the time dependency of sequential decisions when migrating the task from one server to another. In [25], the authors express the SPP as a contextual MAB problem. They can decouple the time dependency using the contextual feature vector to describe user behavior information. Furthermore, they incorporate user preference into the placement decision-making to achieve better personalized service performance.

c. *Works using other techniques for SPP:* In [23], the authors aim to maximize the QoS for end-users by studying joint service placement and model scheduling of edge intelligence (EI) services. They formulate the problem as an integer linear program. In [22], the authors focused on minimizing computation time and energy consumption for end-users and edge nodes using a mixed-integer non-linear programming problem. They considered end-users and edge nodes' CPU frequencies and uplink bandwidth to an optimal service placement. They presented analytical expressions that enable low-complexity calculation of the most efficient resource allocation decisions. Without requiring any future user mobility as prior knowledge, several works, such as [24], utilize Lyapunov optimization to address the SPP by treating it as an online queue stability control problem.

2) Proactive Mobility Management Approaches: Other works propose to deal with end-user mobility by providing a proactive approach that predicts selected QoS metrics and executes migration service once a QoS degradation is predicted. However, understanding the movement behavior of end devices of edge nodes may be helpful for efficient service placement. Thus, several works predict user traffic to proactively migrate the service based on the predicted trajectory. The surveyed literature is classified, namely, regarding the technique used.

a. *Works using Machine learning for SPP:* Authors in [21] propose a framework based on Mobility-Aware Deep Reinforcement Learning to handle end-users mobility and consider the latency constraint. By adapting Machine learning techniques, reference [20] tackles the

[1]In this context, 'dynamic' does not refer to the network; instead, it indicates a non-static placement.

problem by developing traffic prediction tools that utilize statistical, rule-based, or deep machine learning methods. The authors apply user-traffic prediction to enable proactive resource management and optimize service placement for increased efficiency.

b. *Works using Metaheuristic for SPP:* Authors in [18] and [19] address the problem by proposing a two-time-scale framework that jointly optimizes service placement and request scheduling under storage, communication, computation, and budget constraints. Data placement needs to be adapted over time to serve time-varying demands while considering system stability and operation cost. Like in [16], researchers propose a predictive service placement in Mobile Edge Computing based on user mobility. They study predictive service placement with a limited short-term prediction based on a T-slot algorithm that employs a frame-based design to predict user mobility. To achieve this, they utilized the two-timescale Lyapunov optimization method. Similar to [16], the authors in [17] suggest a service placement algorithm that integrates user mobility prediction using a frame-based design.

c. *Works using statistical models for SPP:* Authors in [11] propose a service migration and resource manager for Vehicular Edge Computing that ensures the necessary vehicle resources. Authors in [15] introduce an intelligent service migration and resource management algorithm in edge-enabled environments that considers the latency constraint. In both works, the ARIMA model is used for prediction.

B. Edge Dynamicity Without Mobility Management in an Edge Computing Context

This section addresses the dynamic nature of the edge infrastructure, which does not account for the mobility pattern of edge nodes. The edge network exhibits high dynamism due to entities frequently joining and departing from the network. In [7], the authors attribute this dynamism to factors such as network link instability, failures, and resource capability variations. Consequently, the challenge lies in devising a placement strategy to install, replace, or remove services efficiently while meeting QoS constraints. In [14], the authors aim to minimize resource costs and fulfill QoS requirements through dynamic service provisioning in the fog infrastructure. At the same time, in [13], a strategy is proposed to dynamically identify host nodes and manage unexpected changes in deployed component frequencies.

C. The Paper's Contributions

Due to the substantial mobility of certain end-users and edge nodes, solutions must ensure continuous service availability and uninterrupted request fulfillment. Services need the capability to migrate across diverse edge nodes in sync with the mobility of end-users/edge nodes. While a limited number of works have addressed the mobility of end-users/edge nodes in their solutions, it is worth noting that most migration approaches focus solely on end-users' mobility. Additionally, the majority of proposed placement techniques that support mobility are reactive. Therefore, adopting a proactive approach to SPP in edge computing and forecasting potential declines in one or more QoS metrics may offer effective service placement and management advantages.

TABLE I
CLASSIFICATION OF PROACTIVE MOBILITY MANAGEMENT IN LITERATURE REFERENCES

	$(REQ1)$ Dynamicity	$(REQ2)$ Mobility	$(REQ3)$ Recovery
[16] [19]	✗	✗	-
[20]	✗	✗	✓
[17] [18] [21]	✓	✗	-
[11]	✓	✗	✓
[15]	✓	-	✓
This paper	✓	✓	✓

The work presented in this paper aligns with the previously described approach, employing a proactive method rooted in statistical Exponential smoothing models to forecast latency-related QoS degradation.

Table I provides a condensed summary of literature references that adopt proactive mobility management strategies based on the requirements outlined in Section II-D. As a reminder, the requirement (*REQ*1) pertains to the dynamic evolution of the network topology, (*REQ*2) addresses the potential mobility of edge nodes and end-users, and (*REQ*3) focuses on the capacity to rebound from possible degradation in prediction quality.

References [16], [19], [20] do not address network dynamicity and mobility of end-users and edge nodes. Authors in [20] guarantee recovery explicitly. References [17], [18], [21] consider network dynamicity but do not account for the mobility of end-users and edge nodes. It is unclear whether they address recovery through degradation prediction. Authors in [11] and [15] tackle network dynamicity and guarantee recovery but do not address the mobility of end-users and edge nodes. This paper distinguishes itself by assuming network dynamicity, focusing on QoS latency degradation, and addressing the mobility of end-users and edge nodes, including recovery through degradation prediction.

IV. Background and Fundamental Concepts

This section briefly introduces the concept of time series, including its definition and core components, and presents a selection of prediction models employed for univariate time series forecasting.

A. Time Series

A set of recorded observations x_t, each corresponding to a specific time t, can be referred to as a time series (TS) [30]. A time series can be univariate or multivariate. It is called univariate when it represents a single set of collected observations ordered in time. In this study, a discrete univariate time series is considered.

B. Time Series Decomposition

According to [31] and [32], a time series may include a possible:

- Trend (fall or rise in the mean): T_t
- Seasonality (recurring cycle): S_t
- Remaining random Residual: R_t

Depending on its model, i.e., whether it is additive, multiplicative, or combined [33], the TS can be considered as follows:

- Additive: $y_t = S_t + T_t + R_t$
- Multiplicative: $y_t = S_t * T_t * R_t$
- Combined: $y_t = (S_t + T_t) * R_t$

The way to extract a component from a given TS is called decomposition. This decomposition allows the separation of the TS data based on its core components for different purposes.

C. Prediction Models for Time Series

There are different prediction methods for univariate time series. For example, linear methods like Exponential smoothing, ARIMA and its variants, and non-linear methods like Markov Switching [34] or ARCH models.

The choice was made to use Exponential smoothing models due to their proven efficiency in online and short-term time series forecasting, as demonstrated in previous studies [35], [36], in addition to their simplicity [37]. ARIMA is used as the benchmark model due to its robust forecasting performance and low complexity [11].

1) Exponential Smoothing: Exponential smoothing methods use weighted averages for time series data, emphasizing recent observations. The statistical models for these methods are state space models and referenced as *ETS* for (Error, Trend, Seasonal), each featuring measurement and state equations for observed and unobserved components (e.g., level, trend, seasonal). Both models come in two variations: additive and multiplicative errors, producing the same point forecasts with matching smoothing parameter values. This paper focuses on two methods: Simple Exponential Smoothing (SES) for stationary time series, i.e., a TS with time-independent statistical properties [38]; and Holt's Damped Linear Trend method [39], [40] for non-stationary data. Given the exclusive interest in the point forecast, the model with additive (A) errors is utilized, making 'E' in the adopted ETS models stand for 'A'.

a. *Simple Exponential Smoothing model:* A simple exponential smoothing model with additive errors, or $ETS(A, N, N)$, can be written as [38]:

$$y_t = l_{t-1} + \varepsilon_t \tag{1}$$
$$l_t = l_{t-1} + \alpha\varepsilon_t \tag{2}$$

Equations 1 and 2 are referred to as the measurement equation and state equation, respectively, where: y_t is the current observation, l_{t-1} is the previous series level, i.e., the predictable portion of y_t, ε_t is the random error, l_t is the adjusted level, and α is the smoothing parameter.

b. *Damped Holt's linear model:* A damped Holt's linear method with additive errors, or $ETS(A, A_d, N)$, can be written as [38]:

$$y_t = l_{t-1} + b_{t-1} + \varepsilon_t \tag{3}$$
$$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t \tag{4}$$
$$b_t = b_{t-1} + \alpha\beta\varepsilon_t \tag{5}$$

where: y_t, l_{t-1}, ε_t, l_t, and α remain the same as in equations (1) and (2). b_t estimates the trend at time *t*, and β is the trend's smoothing parameter.

2) ARIMA: ARIMA models may represent different types of time series, e.g., exclusive autoregressive (AR), which tries to predict the future values of the TS based on its own *p* past behavior, i.e., lagged values; exclusive moving average (MA), which tries to predict the future values of the TS based on its own *q* past forecast errors; and mixed AR and MA (ARMA) series. The integrated part *I* of order *d* represents the number, i.e., *d*, of the differences needed to make the TS stationary.

An ARIMA model of orders *p*, *d*, and *q*, i.e., $ARIMA(p, d, q)$ can be written as [38]:

$$y\prime_t = c + \phi_1 y\prime_{t-1} + \cdots + \phi_p y\prime_{t-p} + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q} + \varepsilon_t \tag{6}$$

where $y\prime_t$ is the differenced series, c is the intercept, $\phi_1 \cdots \phi_p$ represent the coefficients of the lagged values, $y\prime_{t-1} y \cdots \prime_{t-p}$ represent the lagged values of $y\prime_t$, $\theta_1 \cdots \theta_q$ represent the coefficients of the past forecast errors, $\varepsilon_{t-1}, \ldots \varepsilon_{t-q}$ represent the past forecast errors and ε_t represents the white noise.

V. Model Design and Workflow Overview

The vehicle's migration engine requires knowledge of the appropriate edge node for migrating the decision module. For this purpose, a predictive approach is pursued, involving creating a prediction service. This service assesses edge nodes based on specific criteria, with latency being the QoS metric. The edge node with the lowest latency is selected for hosting the decision module. Subsequent sections detail the prediction service and various techniques and combinations to improve the forecasting process.

A. Workflow Overview

The objective is to forecast latency values based on historical data. This process, depicted in Figure 3, comprises two primary stages, each consisting of several phases.

The first stage, the *Preparation stage*, encompasses the initial four phases. Its purpose is to collect TS data and ensure their stationarity while identifying and addressing outliers.

The second stage, the *Prediction stage*, involves the remaining two phases. Its goal is to identify an appropriate model for predicting new latency values and assess the quality of these predictions. This entire process is repeated for each incoming latency value.

B. Algorithms & Flowcharts

This section outlines the stages and phases presented in the workflow overview of Figure 3, delving into the algorithms and approaches used in each.

1) Preparation Stage:

a. *Time series build:* The first phase involves constructing the TS by collecting latency values. A minimum

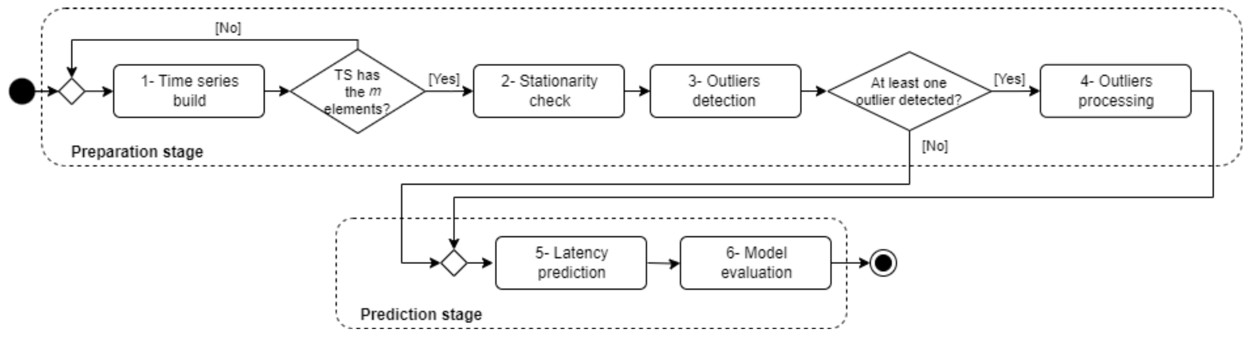


Fig. 3. Workflow overview.

threshold number of values, denoted as *m*, is necessary to initiate the forecasting process. Once this threshold, *m*, is met, the process proceeds to the stationary check phase.

i. *Time series size:* Two forecast sizes are tested: cumulative and sliding window.
   - *Cumulative size:* All the gathered latency values are considered in the prediction process.
   - *Sliding window size:* Only the most recent part of the TS, i.e., a *window* of *w* latency values, is considered, with $w \geq m$.

b. *Stationarity check:* The stationarity condition influences the selection of the forecast model and its parameters. The augmented Dickey-Fuller (ADF) unit-root test is utilized to verify the TS stationarity, ensuring that its statistical properties remain constant over time.
Under the assumption of a normal distribution, if the TS is stationary, the *ETS*(*A*, *N*, *N*) model is used; otherwise, the $ETS(A, A_d, N)$ model is chosen.

c. *Outliers' detection:* An outlier, as explained in [41], is an observation that significantly deviates from others, suggesting a distinct mechanism might have generated it. This deviation, as discussed in [38], implies that the observation has an extreme value compared to other homogeneous observations in the time series (TS) [42]. Outliers can represent anomalies, errors, or rare events, and their nature can be assessed by experts in some cases.
This work utilizes automated outlier detection, testing two scopes and techniques to identify the most suitable approach.

i. *Outliers' detection scope:* The following subsections describe two utilized scopes: detecting outliers across all time series or within their residual components.
   - *All the time series:* This scope consists of detecting the outliers of the given time series without a decomposition phase, i.e., considering the TS as a whole.
   - *Time series' residuals:* This scope involves outlier detection based on the residual component of the time series. The time series is decomposed into its core components, as detailed in Section IV-B, focusing on the residual component R_t for outlier detection. The choice of decomposition depends on the nature of the time series, which can be either additive or multiplicative. This paper assumes that the time series follows either an additive or multiplicative pattern, and experiments are conducted to identify this pattern computationally.

ii. *Outliers' detection technique:* For both detailed scopes, two techniques are used for the detection of outliers: the first one is the Boxplot technique, and the second one is the moving mean or rolling mean technique. These techniques are detailed in the following subsections.
   - *Boxplot technique:* Boxplot is a statistical and data analysis tool to study distribution characteristics and detect outliers [42]. Even if Boxplot is a "visual" technique, it can be computed, and results can be automatically retrieved.
   - *Moving mean technique:* Moving mean or rolling mean technique, with an adapted 3σ rule criterion, also called *"three standard deviations from the mean,"* [43] [44] can be used to detect outliers. It considers two thresholds: a minimum and a maximum. If a value is outside those thresholds, it is considered an outlier. The minimum threshold thr_{min} and the maximum thr_{max} may be computed respectively by equations 7 and 8 as follows:

$$thr_{min} = mean - p * StandardDeviation \tag{7}$$

$$thr_{max} = mean + p * StandardDeviation \tag{8}$$

The parameter *p* refers to the width of the moving mean. After some experimentation, *p* is fixed to 2.

If at least one outlier is detected, the process moves to the outliers processing; otherwise, the process moves directly to the prediction phase.

d. *Outliers processing:* When detected, the choice is made on whether the outliers are removed or replaced by the median.

TABLE II
CONDUCTED PREDICTION CONFIGURATIONS

			Configurations				
			C1	C2	C3	C4	C5
TS size		Sliding window	✓	✓			
		Cumulative			✓	✓	✓
Outliers	Detection scope	TS	✓	✓	✓	✓	
		Residual					✓
	Detection technique	Box Plot	✓				
		Moving Mean		✓	✓	✓	✓
	Processing operation	Replacement	✓	✓	✓		
		Removal				✓	✓

2) Prediction Stage:

a. *Prediction:* Based on the stationary check results, an exponential smoothing model is applied for a one-step-ahead forecast.

b. *Model evaluation:* The predicted latency value is compared to the real value, and a set of evaluation criteria, i.e., Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE), are processed based on these values. These metrics help select the most suitable approach for the paper's use case.

VI. MODEL IMPLEMENTATION AND EVALUATION

This section builds upon the techniques discussed in the previous section regarding the prediction service. It presents and discusses the results of tested prediction configurations, including various outlier detection and processing methods.

A. Configuration-Based Prediction Setup

The simulation scenarios ran on a single machine equipped with an Intel Core i5-4210U CPU @ 1.70GHz 2.40 GHz processor, 12 GB RAM, and Windows 10 (64-bit). Using Python, the statsmodel library [45], which supports *ETS*(*A*, *N*, *N*) and $ETS(A, A_d, N)$, and Jupyter Notebook as the simulation environment, a series of experiments were conducted.

The dataset under consideration was generated in a prior study involving an operational autonomous car prototype introduced in [12]. The dataset comprises 10,000 rows, with a 1-second interval between each measurement. Each row provides information on the latency between the car and a neighboring edge node.

The experiments conducted on the dataset involved altering the TS size and the scopes, techniques, and policies used for detecting and processing outliers. Each distinct variation is referred to as a 'configuration.' The most relevant configurations are listed in Table II. For example, in the first configuration, $C1$, the outliers' detection is made directly on the TS, i.e., without decomposing the TS, using the Moving Mean technique and with a replacement policy. The prediction is made considering a sliding window size.

QoS latency predictions are conducted with different configurations, and results are compared to detect the most suitable one for the case study. The subsequent sections contain the presentation of the findings.

TABLE III
AVERAGE EXECUTION TIME SUMMARY FOR EACH CONFIGURATION-BASED PREDICTION

Configuration	Average execution time (second)
C1	0.087842
C2	**0.086641**
C3	0.157405
C4	0.149422
C5	0.158219

TABLE IV
VALIDATION CRITERIA SUMMARY FOR EACH CONFIGURATION-BASED PREDICTIONS

Configuration	RMSE	MSE	MAE
C1	$2.7746 * 10^{-2}$	$0.0769 * 10^{-2}$	$2.4025 * 10^{-2}$
C2	$2.7752 * 10^{-2}$	$0.0770 * 10^{-2}$	$2.4025 * 10^{-2}$
C3	$2.7352 * 10^{-2}$	$0.0748 * 10^{-2}$	$2.3730 * 10^{-2}$
C4	$2.7352 * 10^{-2}$	$0.0748 * 10^{-2}$	$2.3730 * 10^{-2}$
C5	$\mathbf{2.7350 * 10^{-2}}$	$\mathbf{0.0748 * 10^{-2}}$	$\mathbf{2.3726 * 10^{-2}}$

B. Configuration-Based Prediction Results and Discussion

This subsection presents the results for execution time and evaluation metrics of the configuration-based prediction. It includes comparisons of selected configurations, where one parameter is altered at a time while keeping the others fixed. The section also provides an interpretation of the results.

1) Configuration-Based Prediction Results:

a. *Execution time:* The average execution time of each configuration-based prediction is presented in Table III. Configuration C2-based predictions show an advantageous execution time.

b. *Evaluation metrics:* RMSE, MSE, and MAE are calculated for each configuration-based prediction and resumed in Table IV below. The values are truncated to 4 decimal places. Configuration C5-based predictions show favorable accuracy values.

c. *Configuration-based prediction comparisons:* This section compares selected configurations to determine the most suitable parameters for the current case study. Each configuration parameter from Table II is compared by evaluating two configuration-based prediction. These predictions are plotted alongside the real latency values of the dataset. Each figure includes three plots:
   - One for the real latency values labeled as *'Real'* and is represented by a solid blue line,
   - Two for the configuration-based prediction labeled as *'ETS Predictions C*'*, where ETS refers to the prediction model and (*) refers to the configuration number in Table II. Dashed lines represent these predictions.

   The comparison also relies on the results presented in Tables III and IV, which, respectively, illustrate the execution time and evaluation metrics associated with configuration-based prediction.

   i. *Computed element size:* In this scenario, a comparison occurs between two configurations: C2, employing the sliding window size, and C3, utilizing the cumulative size.

   Figure 4 displays prediction plots for C2 (in green) and C3 (in pink) alongside the real latency data (in blue). These plots more effectively replicate

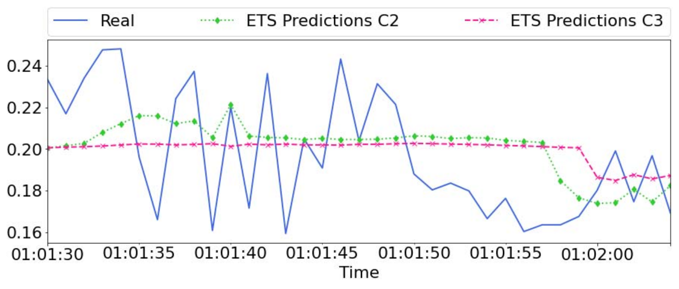


Fig. 4. Computed element size: Sliding window (C2) vs. Cumulative (C3).

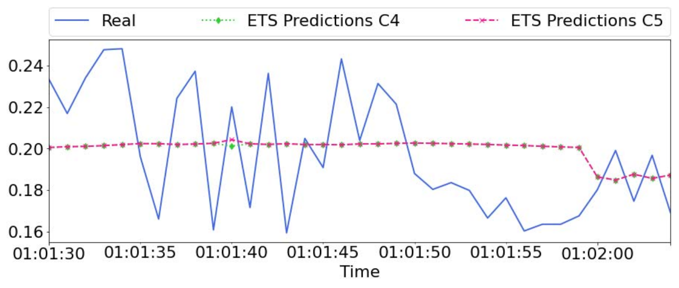


Fig. 5. Outliers' detection scope selection: TS (C4) vs. Residual (C5).

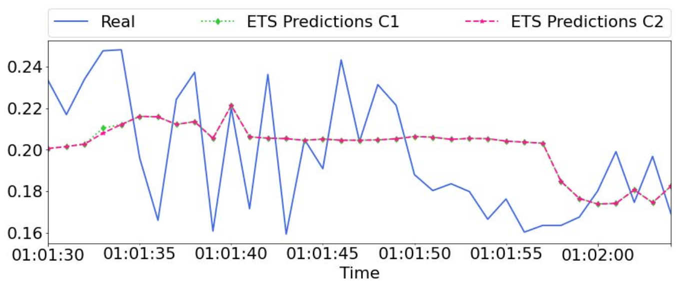


Fig. 6. Outliers' detection technique selection: Box Plot (C1) vs. Moving Mean (C2).

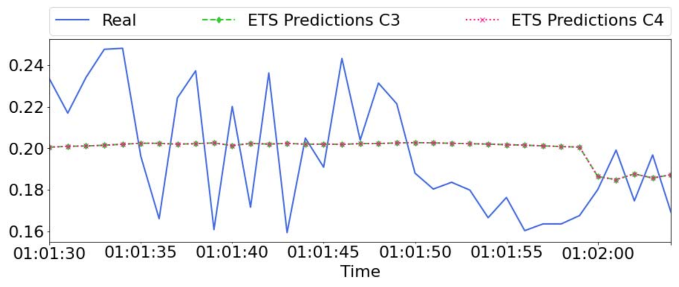


Fig. 7. Outliers processing operation selection: Replacement (C3) vs. Removal (C4).

the fluctuations in the real data, particularly for C2. Regarding evaluation metrics, C3 has slightly lower values than C2. However, C2 demonstrates an advantage in computational efficiency, with a lower average processing time than C3.

Considering the trade-off between accuracy and efficiency in a real-time context, C2 is slightly preferred over C3, making the *sliding window size* the recommended option.

ii. *Outliers' detection scope:* In this scenario, a comparison is made between two configurations: C4, which operates on the time series, and C5, which utilizes the residual part of the time series.

Figure 5 presents prediction plots for C4 (in green) and C5 (in pink), along with the actual data (in blue). These plots more accurately capture the fluctuations in the real data, particularly for C5. Regarding evaluation metrics, C5 has slightly lower values than C4. However, C4 has an advantage in computational efficiency, featuring a lower average processing time than C5.

In this context, choosing C4 is preferable over C5, making the time series *decomposition not* the recommended option.

iii. *Outliers' detection technique:* In this scenario, a comparison occurs between two configurations: C1, employing the Boxplot technique, and C2, utilizing the moving mean technique.

Figure 6 displays prediction plots for C1 (in green) and C2 (in pink) alongside the actual data (in blue). These plots more effectively replicate the fluctuations in the real data, particularly for C1. This observation finds support in the RMSE values, where C1's RMSE is slightly lower than that of C2. Additionally, regarding computational efficiency, C2 exhibits a slight advantage with a lower average processing than C1.

Considering the balance between accuracy and speed, C2 emerges as a slightly superior choice to C1, making the *moving mean technique* the preferred option.

iv. *Outliers' processing operation:* In this context, a comparison is drawn between two configurations: C3, replacing outliers with the median value, and C4, removing outliers.

Figure 7 presents prediction plots for C3 (in green) and C4 (in pink) alongside the actual data (in blue). These plots equally replicate the fluctuations in the real data. Regarding evaluation metrics, both C3 and C4 have identical values. However, C4 exhibits an advantage in computational efficiency, with a lower average processing time compared to C3.

C4 can be a preferred choice over C3, making the *removal* of the outliers the recommended option.

2) Discussion: In the applied use case and from this experiment, an effective latency prediction model, coupled with outlier detection and processing, would involve the sliding window size. Outliers would be detected without time series decomposition, employing the moving mean technique for detection and removal processing.

VII. Proof-of-Concept

This section introduces the prototype validating the paper's contributions, namely the proposition of a QoS degradation prediction model (*CONTR*1) and a proactive decision module placement based on QoS degradation prediction (*CONTR*2).

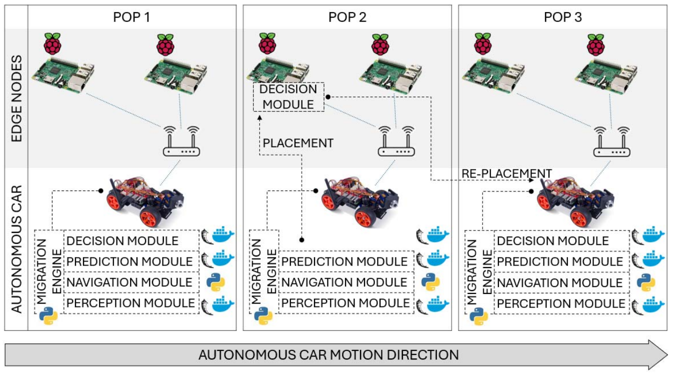


Fig. 8. Prototype architecture.

TABLE V
SUMMARY OF RESULTS OF VALIDATION CRITERIA FOR ETS AND ARIMA MODELS

Model	Outliers detection	RMSE	MSE	MAE
ETS	✓	$\mathbf{2.7283 * 10^{-2}}$	$\mathbf{0.0744 * 10^{-2}}$	$\mathbf{2.4244 * 10^{-2}}$
	✗	$4.0926 * 10^{-2}$	$0.1674 * 10^{-2}$	$2.9572 * 10^{-2}$
ARIMA	✓	$2.8263 * 10^{-2}$	$0.0798 * 10^{-2}$	$2.4775 * 10^{-2}$
	✗	$6.6680 * 10^{-2}$	$0.4446 * 10^{-2}$	$3.6078 * 10^{-2}$

TABLE VI
ETS AND ARIMA EXECUTION TIME STATISTICS

	ETS	ARIMA
Maximum	**0.047**	4.375
Minimum	**0.001**	0.297
Mean	**0.018**	1.441

A. Implementation Details

A global overview of the prototype architecture is depicted in Figure 8.

The configuration of the prototype involves:

- a Raspberry SunFounder PiCar-V representing the autonomous car,
- a Raspberry Pi 3 Model B (Quad Core CPU 1.2 GHz, 1 GB RAM), and a Raspberry Pi 2 Model B (Quad Core CPU 1.2 GHz, 1 GB RAM) devices acting as edge computing nodes.

The car and the edge nodes communicate through Wi-Fi. Flask,[2] Postman,[3] Putty,[4] Docker,[5] and DockerHub[6] are employed to develop and test Dockerized Flask Web applications representing the car's prediction, decision, and perception modules. The code related to these modules can be found in GitHub.[7] The prediction module uses the statsmodel [45] and pmdarima [46] libraries, which support $ETS(A, N, N)$ and $ETS(A, A_d, N)$, and *ARIMA*, respectively.

B. Experiment Results and Lessons Learned

This subsection presents and explains the experimental results contributing to the lessons learned.

1) Experiment Results:

a. *Prediction model:* Utilizing the dataset from the preceding Section IV-A, the prediction module, employing ETS models, selects a suitable edge node with the ARIMA model as a benchmark. Experiments validate the chosen model and the outliers' detection and processing phases. Two sets of predictions are made—one includes detecting and processing outliers before the prediction process, and the other omits this step. The comparison is based on accuracy and execution time.
Table V shows the RMSE, MSE, and MAE for latency predictions from ETS and ARIMA models, with and without outlier detection and processing phases. Values are rounded to four decimal places.
Table VI provides execution time statistics for the models.
Both models perform similarly in terms of accuracy, exhibiting improved behavior when the outlier detection and processing phases are included in the prediction process. However, in terms of execution time, the ETS models demonstrate superior performance.

b. *Proactive decision module based on QoS degradation prediction:* Based on the QoS latency degradation prediction model discussed in Section VI and the preceding subsection, the proactive decision module initially selects an appropriate edge node to host the car's decision module. It replaces the decision module in an alternate edge node upon detecting predicted latency degradation. If unavailable, the module shifts operation within the car, ensuring continuous and uninterrupted functionality. Within the framework outlined in Section VII-A, the Raspberry SunFounder PiCar-V collects latencies from two Raspberry Pi units while in motion, using the *ping* command. A prediction is generated for each unit, and the one with the lowest latency is chosen. At the experiment's outset, the decision module operates uniformly across all nodes for simplicity. When an edge node is chosen, its associated decision module is accessed as necessary. Latency predictions are computed every 20 seconds to detect degradation; if identified, the decision module on the car is activated. The process of identifying a suitable edge node is then reiterated.

2) Lessons Learned: The utilization of ETS models demonstrated superior overall efficiency compared to the ARIMA model, with improved predictions through outlier detection and processing phases.

In this hypothetical use case, placing the proactive decision module in the chosen edge node enables the detection of QoS degradation beforehand, ensuring secure and continuous car operation. This proactive approach, detailed in Section II, enables the prediction of QoS degradation.

Regarding the support of the requirements outlined in Section II-D, the paper's contributions, defined in the introduction (Section I), meet these requirements:

- (*REQ*1) and (*REQ*2) are addressed by using a dataset containing latencies of moving car and edge nodes in the simulation (Section VI-A) and in the prototype (Section II).

[2]https://flask.palletsprojects.com/en/3.0.x/
[3]https://www.postman.com/
[4]https://putty.org/
[5]https://www.docker.com/
[6]https://hub.docker.com/
[7]https://github.com/HendaSfx/raspModules

- (*REQ*3) is addressed by (*CONTR*1) and (*CONTR*2), as the system is designed for recovery.

VIII. Conclusion and Future Work

This paper proposes a proactive service (re)placement approach based on the prediction of QoS degradation. The degradation prediction relies on Exponential Smoothing methods to determine a suitable edge node for placing the car's decision module, with the selection based on minimizing latency. Multiple outlier detection configurations were evaluated.

A proof-of-concept validates the chosen model, - by comparing it to ARIMA-, and the proposed proactive service (re)placement approach. This approach ensures the continuity of the placed module, suggesting the feasibility of locating non-critical modules on edge nodes.

The next step involves comparing Exponential Smoothing models, a deep learning model, and a hybrid model (Exponential Smoothing + Deep Learning) for improved prediction. Subsequently, additional criteria for QoS degradation prediction, such as the accessibility of resources (CPU/GPU capacity), availability, and the battery level of edge nodes, will be considered.

References

[1] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020.

[2] K. L. Lueth. "State of IoT." Accessed: Nov. 19, 2023. [Online]. Available: https://iot-analytics.com/wp/wp-content/uploads/2023/05/Insights-Release-State-of-IoT-2023-Number-of-connected-IoT-devices-growing-16-to-16.0-billion-globally.pdf

[3] "IoT connections outlook." Ericsson. 2021. Accessed: May 13, 2022. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/iot-connections-outlook

[4] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[5] Z. Wang, Z. Zhong, and M. Ni, "A semi-Markov decision process-based computation offloading strategy in vehicular networks," in *Proc. IEEE 28th Annu. Int. Symp. Personal, Indoor, Mobile Radio Commun. (PIMRC)*, 2017, pp. 1–6.

[6] L. Gao, T. H. Luan, B. Liu, W. Zhou, and S. Yu, "Fog computing and its applications in 5G," in *5G Mobile Communications*. Cham, Switzerland: Springer, 2017, pp. 571–593.

[7] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Comput. Surveys*, vol. 53, no. 3, pp. 1–35, 2020.

[8] H. Yu, H. Lee, and H. Jeon, "What is 5G? Emerging 5G mobile services and network requirements," *Sustainability*, vol. 9, no. 10, p. 1848, 2017. [Online]. Available: https://www.mdpi.com/2071-1050/9/10/1848

[9] B. Blanco et al., "Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN," *Comput. Stand. Interfaces*, vol. 54, pp. 216–228, Nov. 2017.

[10] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Aug. 2019.

[11] L. S. Pacheco, D. L. Rosário, E. C. Cerqueira, and L. A. Villas, "Service migration in edge computing environments for connected autonomous vehicles," in *Proc. Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2020, pp. 533–546.

[12] F. Raissi, S. Yangui, and F. Camps, "Autonomous cars, 5G mobile networks and smart cities: Beyond the hype," in *Proc. IEEE 28th Int. Conf. Enabling Technol. Infrastruct. Collaborative Enterprises (WETICE)*, 2019, pp. 180–185.

[13] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, no. 9, 2018.

[14] A. Yousefpour et al., "QoS-aware dynamic fog service provisioning," 2018, *arXiv:1802.00800*.

[15] L. Pacheco, D. Rosário, E. Cerqueira, L. Villas, T. Braun, and A. A. Loureiro, "Distributed user-centric service migration for edge-enabled networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, 2021, pp. 618–622.

[16] H. Ma, Z. Zhou, and X. Chen, "Leveraging the power of prediction: Predictive service placement for latency-sensitive mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6454–6468, Oct. 2020.

[17] J. Plachy, Z. Becvar, and E.-C. Strinati, "Dynamic resource allocation exploiting mobility prediction in mobile edge computing," in *Proc. IEEE 27th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, 2016, pp. 1–6.

[18] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 1459–1467.

[19] V. Farhadi et al., "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, Apr. 2021.

[20] A. Azari, P. Papapetrou, S. Z. Denic, and G. Peters, "User traffic prediction for proactive resource management: Learning-powered approaches," 2019, *arXiv:1906.00951*.

[21] C.-L. Wu, T.-C. Chiu, C.-Y. Wang, and A.-C. Pang, "Mobility-aware deep reinforcement learning with glimpse mobility prediction in edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2020, pp. 1–7.

[22] Z. Lin, S. Bi, and Y.-J. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," 2021, *arXiv:2011.05708*.

[23] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," 2021, *arXiv:2104.15094*.

[24] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[25] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 1468–1476.

[26] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2018, pp. 1–7.

[27] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[28] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, "A dynamic service migration mechanism in edge cognitive computing," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–15, 2019.

[29] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *Proc. Global Internet Things Summit (GIoTS)*, 2017, pp. 1–6.

[30] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. New York, NY, USA: Springer-Verlag, 2002, pp. 1–2.

[31] "Time series decomposition & prediction in Python," 2019. Accessed: Aug. 13, 2021. [Online]. Available: https://www.pythonforfinance.net/2019/07/22/time-series-decomposition-prediction-in-python/

[32] A. Mitrani. "Time series decomposition and statsmodels parameters." 2020. Accessed: Aug. 13, 2021. [Online]. Available: https://towardsdatascience.com/time-series-decomposition-and-statsmodels-parameters-69e54d035453

[33] Y. Aragon, *Démarche de Base en Séries Temporelles*. Paris, France: Springer-Verlag, 2011, ch. 1, p. 16.

[34] S. M. Potter, "Nonlinear time series modelling: An introduction," *J. Econ. Surveys*, vol. 13, no. 5, pp. 505–528, 1999.

[35] Z. Dong, D. Yang, T. Reindl, and W. M. Walsh, "Short-term solar irradiance forecasting using exponential smoothing state space model," *Energy*, vol. 55, pp. 1104–1113, Jun. 2013.

[36] S. Mahajan, L.-J. Chen, and T.-C. Tsai, "Short-term PM2.5 forecasting using exponential smoothing method: A comparative analysis," *Sensors*, vol. 18, no. 10, p. 3223, 2018.

[37] J. Köppelová and A. Jindrová, "Application of exponential smoothing models and Arima models in time series analysis from Telco area," *AGRIS On-Line Papers Econ. Informat.*, vol. 11, no. 665-2019-4145, pp. 73–84, 2019.

[38] R. J. Hyndman and G. Athanasopoulos, *ARIMA Models*. Melbourne, VIC, Australia: OTexts, 2018, ch. 8.

[39] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *Int. J. Forecast.*, vol. 20, no. 1, pp. 5–10, 2004.

[40] E. S. Gardner Jr and E. McKenzie, "Forecasting trends in time series," *Manag. Sci.*, vol. 31, no. 10, pp. 1237–1246, 1985.

[41] D. M. Hawkins, *Introduction*. London, U.K.: Chapman Hall, 1980, ch. 1, p. 9.

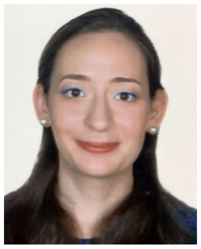
[42] Y. H. Dovoedo and S. Chakraborti, "Boxplot-based outlier detection for the location-scalfamily," *Commun. Stat. Simulat. Comput.*, vol. 44, no. 6, pp. 1492–1513, 2015.

[43] S. Chawla and P. Sun, "SLOM: A new measure for local spatial outliers," *Knowl. Inf. Syst.*, vol. 9, no. 4, pp. 412–429, 2006.

[44] Z. Yao, J. Xie, Y. Tian, and Q. Huang, "Using Hampel identifier to eliminate profile-isolated outliers in laser vision measurement," *J. Sens.*, vol. 2019, Jul. 2019, Art. no. 3823691.

[45] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with Python," in *Proc. 9th Python Sci. Conf.*, 2010, pp. 92–96.

[46] T. G. Smith et al. "Pmdarima: ARIMA estimators for Python." Accessed: 2017. [Online]. Available: http://www.alkaline-ml.com/pmdarima



Henda Sfaxi received the National Engineering Diploma degree in software engineering from the National Institute of Applied Sciences and Technology in 2010, and the M.Sc. degree in software engineering from the Higher Institute of Computer Science, Tunis, Tunisia, in 2012. She is currently pursuing the Doctoral degree with the National School of Engineers of Sfax (ENIS), specializing in computer systems engineering, in the Research laboratory on Development and Control of Distributed Applications (ReDCAD). She is interested in resource management (including migration and offload) of computing and storage resources in distributed systems. Her research interests include edge computing, 5G networks, mobile networks, and vehicular ad hoc networks.

Imene Lahyani received the Diploma degree in computer science and the M.Sc. degree (DEA) from the National School of Engineering of Sfax-Tunisia in 2007 and 2008, respectively, and the Ph.D. degree in computer science from INSA Toulouse and ENIS–Sfax in 2015. She is an Associate Professor with the Department of Computer Science, National School of Engineers of Sfax (ENIS). She is a member of the Research laboratory on Development and Control of Distributed Applications (ReDCAD). Her current research areas include software engineering of distributed systems, self-adaptive systems, and autonomic middleware. She published several scientific papers in high-ranked conferences and journals in her field of research.

Sami Yangui (Member, IEEE) received the M.Sc. degree in computer science from University Tunis-ElManar, Tunisia, in 2010, the Ph.D. degree in computer science from Télécom SudParis, France, in 2024, and the Professorial accreditation (Habilitation) degree from the L'Institut National Polytechnique de Toulouse in 2024. He is an Associate Professor with the Institut National des Sciences Appliquées, Toulouse, France. He is a member of LAAS-CNRS Research Lab. He is involved in different European and International projects, as well as standardization efforts. His research interests include distributed systems and architectures, service-oriented computing, and Internet of Things. He is working on different aspects related to these topics, such as cloud/edge computing, network functions virtualization, and content delivery networks. He published several scientific papers in high-ranked conferences and journals in his field of research. He served on many program and organization committees of International conferences and workshops, as well as, a guest editor in several journals, such as *Future Generation Computer Systems* (Elsevier) and IEEE ACCESS. He is a member of the evaluation committees of the French National Research Agency and the Fonds de Recherche du Québec – Nature and Technologies, Canada.

Mouna Torjmen received the University Accreditation (Habilitation) degree in 2017, and the Ph.D. degree in computer science from the Paul Sabatier University of Toulouse, France, in 2009. She is currently a Professor. In September 2010, she was appointed as an Associate Professor with the Department of Computer Science, National School of Engineers of Sfax (ENIS). She joined the Research laboratory on Development and Control of Distributed Applications (ReDCAD). Her research interests are currently focused on information systems and Machine learning.