mathematics

MDPI

Article

Automatic Path Planning Offloading Mechanism in Edge-Enabled Environments

Dušan Herich *, Ján Vaščák *, Iveta Zolotová and Alexander Brecko

Department of Cybernetics and Artificial Intelligence, Technical University of Košice, Vysokoškolská 4, 042 00 Košice, Slovakia; iveta.zolotova@tuke.sk (I.Z.); alexander.brecko@tuke.sk (A.B.)
* Correspondence: dusan.herich@tuke.sk (D.H.); jan.vascak@tuke.sk (J.V.)

Abstract: The utilization of edge-enabled cloud computing in unmanned aerial vehicles has facilitated advances in autonomous control by employing computationally intensive algorithms frequently related to traversal among different locations in an environment. A significant problem remains in designing an effective strategy to offload tasks from the edge to the cloud. This work focuses on creating such a strategy by employing a network evaluation method built on the mean opinion score metrics in concoction with machine learning algorithms for path length prediction to assess computational complexity and classification models to perform an offloading decision on the data provided by both network metrics and solution depth prediction. The proposed system is applied to the A* path planning algorithm, and the presented results demonstrate up to 94% accuracy in offloading decisions.

Keywords: edge and cloud computing; k-nearest neighbors; logistic regression; offloading; path planning; UAV

Citation: Herich, D.; Vaščák, J.; Zolotová, I. ; Brecko A. Automatic Path Planning Offloading Mechanism in Edge-Enabled Environments. *Mathematics* **2021**, *9*, 3117. https://doi.org/10.3390/math9233117

Academic Editors: Stefano Carrino, Vicente Rodríguez Montequín, Hatem Ghorbel and Ivana Budinská

Received: 3 November 2021
Accepted: 1 December 2021
Published: 3 December 2021

1. Introduction

The recently perceived emergence of unmanned aerial vehicles (UAVs) with diverse modes of transport: on road, in air, and in water, has facilitated their adoption in a variety of applications ranging from freight and human transportation, environment surveillance to rescue operations. The specific applications include biological material delivery [1], human search and rescue operations [2], emergency medicine [3], warehouse operations [4].

Although the discussed deployments might be deemed successful, a plethora of issues still limit the ubiquitous implementation of UAVs. An unmissable constraint of such systems is their computational capability [5]. While algorithms used by vehicles to fulfill their tasks such as simultaneous localization and mapping, motion planning, and path planning tend to be computationally intensive [6,7], the computing power of devices present on-board vehicles is prone to be constrained either by available energy supply or by the physical space demands [8].

To address the limitation mentioned above, several cloud-enabled systems utilizing on-demand and elastic computing resources were created. The utilization of cloud computing resources as they are defined [9] by the National Institute of Standards and Technology (NIST) have the following characteristics:

- On-demand self-service to enable the unilateral provision of services automatically, without the requirement for human interaction;
- Broad-network access allowing access to cloud capabilities through the network while using standard mechanisms;
- Resource pooling resulting in dynamical assignment and reassignment of resources;
- Rapid elasticity leaving capabilities elastically and rapidly provisionable to reflect the computation requirements;
- Measured service.

The utilization of cloud resources has had immediate implications for the field of autonomous unmanned aerial vehicles, of which the major is increased computational power and storage. Therefore tasks can be performed either on-board a vehicle or offloaded to a remote computing environment, resulting in an expeditious solution of complex assignments such as computer vision, object recognition or mapping by employing highly available and expandable computational resources with the possibility of using parallel computation paradigms. Additionally, the rapid elasticity and on-demand self-service without the requirement for human interaction provided the ability to offload tasks dynamically. Lastly, arising from the nature of cloud computing, access to big data generated by UAV systems was facilitated, while the data exchange became more straightforward as a result of broad-network access. Some examples of cloud systems as discussed are Rapyuta [10], serving as a platform for cloud robotics helping in offloading heavy computation from on-board units by provisioning adaptable environments for cloud computation, the C2TAM cloud framework [11] designed for cooperative mapping and the cloud control system for a group of UAVs [12].

Regardless of the undeniable advantages offered by cloud computing utilization, a significant setback remains the requirement for uninterrupted and expeditious network connection, which in real-world applications cannot be granted at all times. Additionally, in the limited set of scenarios where a stable internet connection can be retained, network-induced traffic elicits from data transfer among the UAV. A remote cloud frequently renders cloud-enabled applications unusable due to the unacceptable response time.

These drawbacks led to the employment of edge-enabled UAV applications. Edge computing transfers computational services to closer proximity with an end-user and, therefore, minimizes the application response time. In comparison with the more traditional cloud computing models, where the application runs on a constrained device while the computationally intensive core services and processes are executed in the cloud environment, the edge-enabled application may operate in three models [13]:

- Cloudlet: may be characterized as a collection of computers disposing of stable internet connectivity, having resources located in close proximity for use by mobile devices. Therefore, it can be regarded as a data center suitable for provisioning resources to end devices in real-time over a WLAN/LAN connection;
- Fog computing: Fog computing resembles a decentralized architecture for computing built upon usually heterogeneous computing nodes, which can be located at various points in the system between the end devices and the cloud environment. The heterogeneity is not transparent for the end device and allocates resources in accordance with requirements;
- Mobile edge computing: Mobile edge computing may be defined as a subset of edge-computing technologies used in a manner bringing computational and storage capabilities to the edge of a radio access network such that the network latency is reduced. The nodes in this implementation are located at the same point as the network controller or a base station.

Application of the edge computing technology in the field of autonomous vehicles can be found in visual navigation, and mapping [14,15], or in drone navigation [16]. While the discussed technology suppresses disadvantages present with both cloud computing and on-board computing, offering a solution to computation and storage constraints, increased latency, and requirement for persistent internet connectivity, an issue yet-to-be addressed is an effective task allocation in order to highlight the advantages of edge-enabled systems further. Therefore, a solution must be developed to improve metrics perceivable in UAV arrangements, such as time to solve a task or energy consumption for a task.

In accordance with the described development of the UAV technologies, this paper is concerned with automatic path planning task scheduling between the edge and the cloud environment. The task of path planning itself is performed over an eight-connected grid-map of various sizes and obstacle densities in concoction with a graph-search algorithm A*. The purpose of the proposed system will be the evaluation of the algorithm runtime by

the means of machine learning, while simultaneously estimating the data transfer time to and from the cloud environment.

Therefore, in regards to the algorithm evaluation, this system will utilize a strategy allowing to omit efforts in exact algorithm runtime prediction, which is derived from the premise that for the purpose of an efficient path planning task offloading between two environments, that is edge and cloud, it is not necessary to perform such prediction, and a dichotomous classification to identify a more advantageous environment for the computation is sufficient.

In order to allow the classification as described, we investigate the behavior of the A* algorithm in grid maps with varying occupancy rates, that is, a differing number of obstacles present in the map. On the basis of the described identification, the system shall predict the length of a potential path plan between two locations on the map. For the purpose of the path length prediction, we propose a regression model, which the system shall provide with a measurement of the map occupancy rate. The prediction of the solution length is deemed to be particularly important for the estimation of the A* algorithm runtime, as it is described in Section 3.

Furthermore, the utilization of the regression model for path length prediction facilitated the employment of classification models to aid offloading decisions. As we have outlined, the purpose of the classification model is a dichotomous classification between the cloud and the edge environment. In order to perform the classification, the model utilized data acquired prior to its deployment. The dataset was built on the measurements of time required to complete a specific problem in each environment, that is, edge and cloud. After the problem was solved in each environment, solution times were compared, and the faster system was recorded alongside the predicted path length. In addition, the dataset includes a mean opinion score measurement as a reflection of comprehensive network quality estimation, which is vital in the offloading consideration due to the fact that an insufficient network quality may significantly prolong the solution time in the case in which a problem is transferred to the cloud. Therefore, the implemented mean opinion score takes several network metrics into consideration, in particular round trip time, data transfer times and packet drop rates, which are demonstrated in later sections to affect response times from the cloud environment or, eventually, inhibit successful solutions.

The described set of capabilities should enable efficient task distribution with the aid of classification models in order to minimize the application response time to the minimum.

Lastly, the designed system will also be constrained by the efficiency and speed of the implemented algorithms and models, as not to mitigate the effects of the scheduling by the slow evaluation speeds. The requirement for speedy evaluation is essential for successful deployment of time-critical applications where performance, availability, and responsiveness are vital for task completion [17].

Ultimately, in this work, we focus on creating a system that can perform an offloading strategy, which would be adaptable in a variety of edge-cloud environments in regards to their performance. The goal is the creation of methods that would not require an extensive supporting infrastructure; however, they would produce near-optimal offloading decisions.

This paper is separated into several sections. The first section deals with a path planning algorithm and analyzes its performance in the edge and cloud environment. Specifically, algorithm runtimes under differing conditions are analyzed and evaluated. In addition, a regression model is created in this section with the aim of assisting in algorithm behavior prediction.

The second section is concerned with computer network performance measurement and evaluation. A set of metrics is described and measured to test their relation to the offloading mechanism behavior. Subsequently, a method for comprehensive network performance evaluation is proposed.

Consequently, the following part focuses on the utilization of a machine learning and network evaluation system to perform dynamic task offloading among the edge and cloud environment. Hence, a dataset on the basis of the proposed metrics is constructed and

supervised machine learning models are defined and trained. Lastly, performance of the designed system is measured and summarized.

2. Related Work

Earlier studies enabling usage of remote computing focused primarily on cloud computing [18,19] mainly in control of unmanned aerial vehicles. However, those systems demonstrated the insufficiency of relying only on cloud computing in real-time. Therefore, several papers focus on latency suppression by designing quality of service-aware [20] and enhanced task assignment among computing environments, especially in collaborative cloud robotics systems [21]. In addition, platforms forming ad hoc cloud computing utilize resources available in other vehicles [22]; however, those resources are sporadically available and unreliable in their nature [23]. For this reason, edge-enabled systems emerged [24,25]. Even though the introduction of edge computing reduced latencies observed in the cloud-only systems and provided available resources to the UAV systems, effective methods to offload some computation tasks from the edge to the cloud had to be established.

Approaches to the edge-cloud computation offloading can be categorized by a stage at which the offloading decision is made—offloading at the design stage or offloading at the runtime stage.

Authors in [26] state that the edge should provide on-demand services, while the cloud should be invoked only when it is necessary. An illustration of system offloading tasks at the design stage can be found in [27], where a simultaneous localization and mapping for indoor mobile robots employ edge–fog–cloud computing architecture designed to segregate tasks among three layers. The lowest layer, called "the robot layer", collects and forwards data towards the upper layers. Some smart city robotics applications [24] separate computation into the device layer, performing only non-demanding computations, an edge layer regarding edge nodes as a hosting infrastructure and a cloud layer performing intensive computational tasks.

In summary, if the decision to offload tasks to the edge is made at the design stage, a pattern where end devices perform only light tasks, edge devices perform more intensive tasks such as collection and analysis of data [28] or hosting a micro datacenter [29], while cloud invocation is performed only for the most resource-demanding problems, such as big data analysis [30] or data streaming [31].

In contrast to the approach to offload tasks at the design stage, systems performing offloading at the runtime were proposed. Similarities in terms of distribution of tasks among the system's components by their demands on computational requirements [32] can be observed. Furthermore, some dynamic offloading strategies focus on energy efficiency, trying to minimize energy consumption during task solving [33,34]. Other classes of dynamic offloading approaches in edge-cloud systems are time-concerned offloading strategies [35] and, respectively, both time and energy-concerned strategies [36].

The offloading decision making can be characterized as a multiobjective optimization problem; therefore, methods may be divided into traditional and advanced offloading decisions [37]. Traditional approaches utilize methods of genetic algorithms [38] or the Stackelberg game algorithm [35]. However, they struggle with the heterogeneity of edge-cloud systems; therefore, their adaptation in differing environments may be troublesome [37]. Conversely, advanced techniques address problems perceived in traditional approaches by introducing deep reinforcement learning [39], blockchain [40], federated learning [41] or recurrent neural networks [42]. Advanced methods may produce near-optimal solutions; however, they require a large enabling infrastructure such as blockchain networks or computer grids intended for deep reinforcement learning [37].

Lastly, offloading mechanisms for the purpose of path planning offloading were developed [43]. Those components were separated into two classes—local and remote. The local planning algorithm is the A*, while the remote one is Dijkstra's algorithm. Primarily, the A* algorithm is used. However, if the system detects a possible improvement

in the selected path, the remote planner is invoked. The decision to offload a task is based on the cost made up of two components—proximity of the trajectory to detected obstacles and the curvature of the trajectory, meaning the system accounts for the cost of path traversal. Hence, the decision omits the availability of the processing capacities regarding the concern of memory requirements.

3. Theoretical Background

The offloading strategy from edge to cloud denotes computation transfer from a device limited in resources to a theoretically resource unconstrained cloud environment. However, cloud computing disposes of vast resources, applications utilizing only cloud struggle with latencies, which results from the requirement to transfer data related to a problem over the internet. In contrast, edge devices are located at the network's edge, closer to the data source. Therefore, latencies posed by data transmission are negligible. The design of offloading strategies and mechanisms is an ongoing research topic, where most work can be split up into two major classes. In the first class, the offloading is secured by the system's architecture. Therefore, the solution is provided with a crisp definition of tasks to be solved on the edge and cloud, respectively. Such implementation may be beneficial when the application consists of more than one module or component. The second class automates the task division between the two. Most of those systems consider a variety of parameters, seeking to find an effective strategy. In most cases, parameters taken into consideration when the offloading strategy is deemed automatic are application runtime, network performance and device performance. Because algorithms for finding path plans are usually not modular, this work focuses on automated offloading strategies. This section investigates path planning algorithm behavior in terms of runtime on devices with different parameters and network performance analysis and evaluation.

3.1. Path Planning Algorithm Runtime Analysis

As indicated earlier, one of the vital parameters to consider when creating an edge-to-cloud offloading strategy is the algorithm's runtime deployed in the system. This work uses the A* algorithm as a reference due to its ongoing widespreadness related to its usability in a wide scope of applications such as efficient path planning in self-reconfigurable robots [44], smart home service systems [45], and automated guided vehicles [46,47].

The algorithm can be regarded as a combination of methods of Dijkstra's search algorithm and heuristic methods such as best-first search used to guide the search, consequently resulting in performance improvement, mainly a reduction in computation time while the algorithm preserves optimality and completeness [48]. A fundamental concept in the discussed algorithm is the evaluation function of the "total cost" containing two parts, defined [49] as follows :

$$f(n) = g(n) + h(n), \tag{1}$$

where $g(n)$ is an actual cost of the optimal path from the starting node to the node under consideration n, while the $h(n)$ resembles the cost from n to a preferred goal node of n.

The heuristic function used in the graph search may significantly affect the algorithm performance [50]. There have been implemented various alterations with varying heuristics in order to reduce time or space complexity [51] and implementations relaxing the admissibility criterion to expedite the process of the path search at the cost of retaining optimality [52]. Further in this work, we employ the Euclidean (straight line) distance as monotonous and admissible in order to retain algorithm optimality:

$$h = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2}, \tag{2}$$

with x_n and y_n denoting coordinates of an evaluated node, while x_g and y_g indicate end node coordinates. The time complexity of the A* algorithm is defined in terms of the branching factor referring to the average number of successors per state, and the solution

depth referring to the shortest possible path as $O(b^d)$ [53] where b is the branching factor and d is solution depth.

To evaluate the algorithm performance in both edge and cloud environments, we have performed a series of tests on randomly generated finite graphs with non-negative weights. A solution was guaranteed to exist on each graph. The first experiment tested runtime dependence on the solution depth. Therefore, the test measured the time the algorithm required to find a solution on a graph with 25% occupied nodes, that is, nontransferable nodes. The trial had 14 levels of graph order. In addition, each step required the algorithm to find a path on ten differing graphs.

Runtime depending on the solution depth is depicted in Figure 1 both for the edge and cloud environments. Figure 2 illustrates the calculated difference in solution times. It can be observed that while there is a minimal time difference between the edge and the cloud when the found path plan is short, for example, on minuscule maps, the difference is more significant when the solution depth increases.

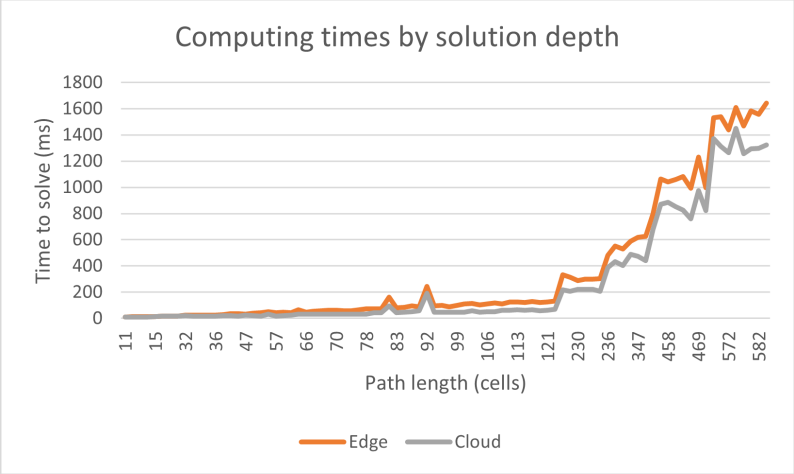

Figure 1. Time required by the algorithm to find a path plan in both edge and cloud environments.

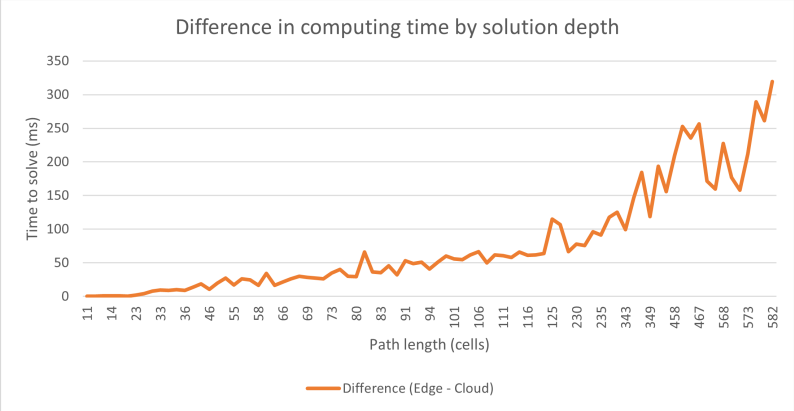

Figure 2. Measured difference in solution times on differently sized maps and different path lengths. It is noticeable that solution time differs insignificantly with smaller maps or shorter paths; however, the difference grows with increasing map and path length.

In pursuance of further comprehension of the path planning algorithm behavior, successive experimental evaluation was performed to examine the relation between the depth of the calculated solution and the rate of cells marked as occupied in the map. On that account, a set of maps with randomly marked occupancy was created. The initial test instance did not contain occupied nodes. Therefore, the occupancy rate was zero. In consecutive test instances, the occupancy rate was increased in steps of 5%. In addition,

the final instance of the test reported an occupancy rate of 30%. The limit as mentioned above in the percentage of cells marked as occupied arose from practical limitations of the random map generator in view of the fact that maps with higher rates were no longer reliably created so that there is a viable path between two selected points.

In addition, every step of the test encompassed ten random maps on which the algorithm was required to calculate a solution corresponding to the earlier experiment. The only exception was the stage with no occupied nodes. During every test stage, the algorithm accomplished one hundred path calculations, that is, ten per every map. Measured outputs were subsequently evaluated by median path length calculation separately for each step of the test. Figure 3 provides an illustration of the results. Graphed values present the median length of the calculated solution as multiples of solutions found in the best-case scenario, meaning on the map with no present obstacles, where the solution would be equal to a straight-line path.

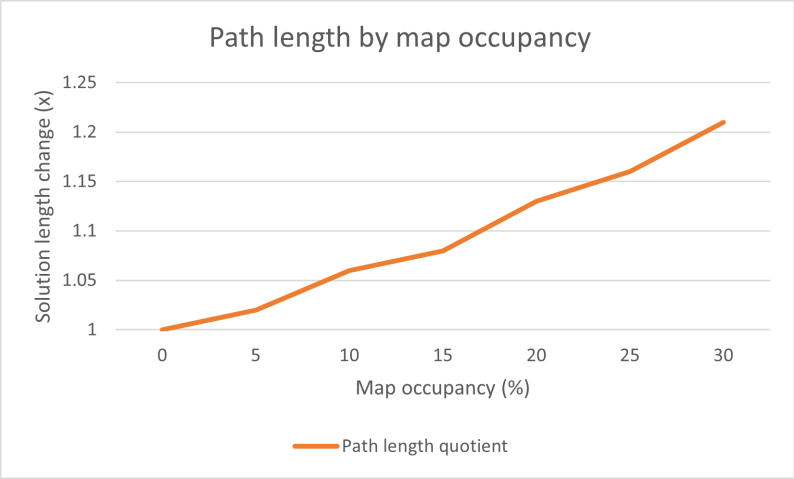

Figure 3. Measured difference in solution times on differently sized maps and different path lengths.

3.2. Network Performance Analysis

Following the definition provided in preceding sections, task offloading from an edge device to a cloud can be in general understood as a strategy to transfer computationally expensive tasks from an edge device frequently constrained in resources to a device, or frequently a set of devices, with an abundance of resources to minimize the time of computation required by a specific task.

It can be concluded that task offloading poses a requirement to transfer the data related to the problem [54,55]. By way of illustration, a map for the path plan calculation has to be transferred from the edge to the cloud environment prior to the path planning algorithm initiation. Subsequently, the established solution must be transferred back from the cloud to the edge, which can be later utilized by a vehicle. In consequence of data transfer requirements, an appropriate offloading strategy should evaluate not only a diminution of computation time but also the performance of the network the data is transferred through should be accounted for [56].

The proposed solution will utilize cloud computing resources available to the general public, which connotes the public cloud paradigm [9]. Public cloud computing resources are in general accessed via public internet connections [57]. Performance of such network connection will typically fluctuate in time [58], with dependence on a number of conditions. This includes the number of applications utilizing the connection at a time, resulting in the alternation of the network bandwidth availability and response times from remote network resources. In case the network uses wireless technologies, the transmission speed may be altered by conditions of the environment changing the signal strength [59].

This section investigates possible approaches to the network performance evaluation necessary to determine the functionality of the application. The evaluation is necessary

as while the network's performance may be sufficient for some non-time-sensitive applications transferring only limited amounts of data, it might be deemed unsatisfactory for an industry-based application demanding a real-time control of machines. The system proposed in this work can be regarded as an instance of such industrial application. Therefore, strict requirements on the performance are posed. Additionally, the designed system can be categorized as telemetry due to the collection of measurements and data with their automatic transmission to other remote points. Performance requirements for such are defined [60] as in Table 1:

Table 1. Network performance requirements for telemetry applications.

Metric	Range
Round trip time	<250 ms
Packet drop rate	0%
Network bandwidth	2 Kbps–50 Mbps

The following sections provide further insights on the specific performance requirements and discuss their impact on the network performance and the application functionality.

3.2.1. Round Trip Time

A round trip time (RTT) resembles an essential metric frequently employed to determine the connection's speed and reliability. The discussed metric represents the amount of time necessary for a signal from one point in the network to reach another point with the addition of time required to acknowledge the signal received. When computer networks are utilized in a solution, a signal whose RTT is measured refers typically to a network packet.

In general, the performance of networked applications increases with reduced RTT [61] by enabling better utilization of a path's capacity, as illustrated in Figure 4 depicting the response times of a cloud server measured on an edge device. For this measurement, the round trip time was artificially increased by the software Clumsy. The graph demonstrates the increase in response times when the RTT time increased even though the server performance, location, used application and protocol were indifferent.

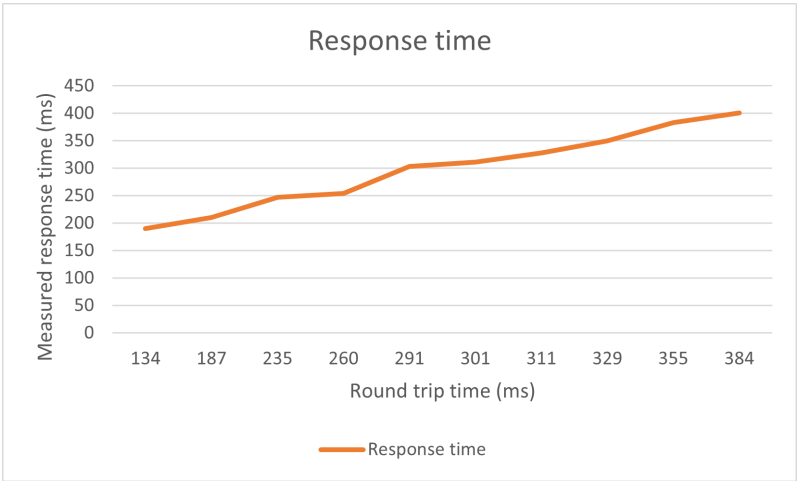

Figure 4. Response time from a cloud server to the edge device in relation to differing round trip times.

3.2.2. Packet Drop Rate

Another evaluated metric is the packet drop rate, representing errors that occurred during data transfer [62]. Precisely, packet drop occurs when packets transferred via a computer network fail to reach their intended destination. Loss of packets may occur on a range of occasions if a network resource is busy and can not deliver data, particularly in

wireless networks if there is radio frequency inference. As a direct result of said loss being detectable, data transmission will take longer due to the behavior of some communication protocols requiring a network resource re-transmitting packets to deliver uncorrupted data. This statement is supported by Figure 5, showing response times from a cloud server measured on an edge device. Similar to the measurements with the round trip time, package loss was induced by a specialized software named Clumsy. Undeniably, response times increased steeply with the growing percentage of lost packets even though server performance, location, used application and protocol went unaltered.

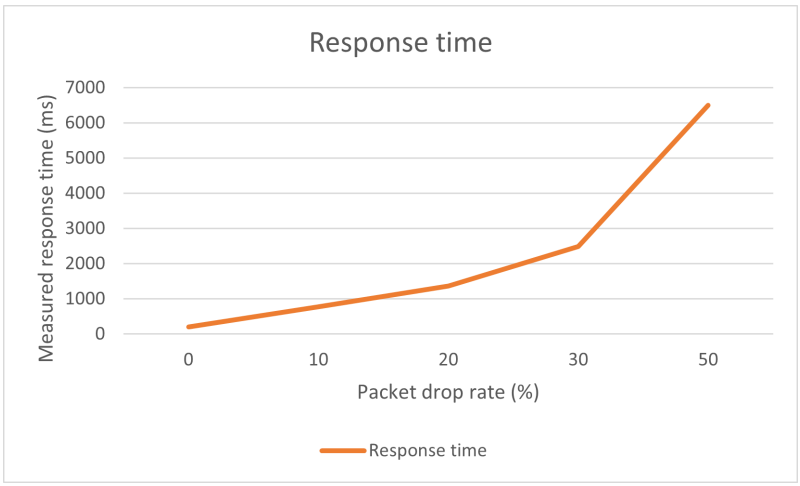

Figure 5. Response times from a cloud server to the edge device in relation to differing packet drop rates.

3.2.3. Data Transfer Time

The last considered metric is the data transfer time signifying the time necessary to transfer data of determining size between two points in the network. The determinate value of said metric can be calculated based on network bandwidth, the maximum rate of data transfer through a given path or based on measured network speed to obtain more accurate results. The calculation is carried out as $T = \frac{I}{R}$ where I refers to data size, while R refers to network speed. Figure 6 depicts measured data transfer times. Further, in this solution, we will use measured network speed to achieve higher accuracy due to defined network bandwidth being typically higher compared to actual speed.

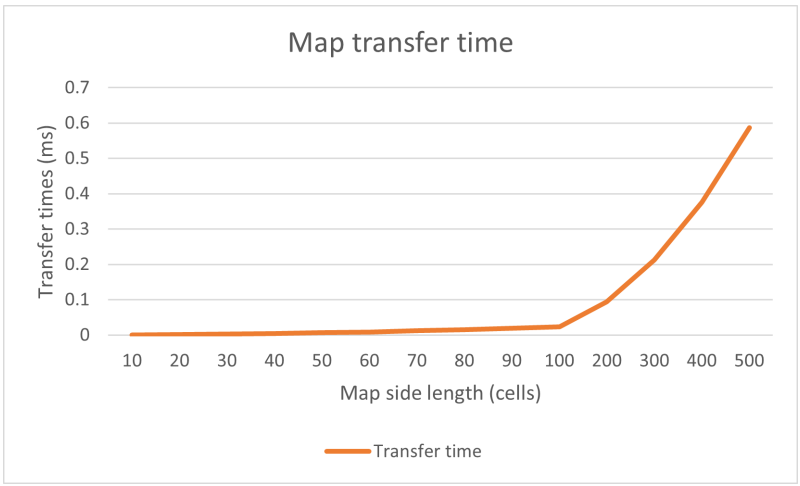

Figure 6. Transfer times of map data. Measurements were made with a connection transmission speed of 1 Gbps.

3.3. Metric Evaluation

In the context of computer networks, metrics are quantitative and qualitative means to determine their desired behavior. There are several various metrics of which evaluation is suitable for disparate use cases, some of which are: capacity, jitter, round trip time, packet loss, and bandwidth or availability [63]. Results of monitoring these parameters are affected by a range of factors, which by way of illustration may be the geographical location of communicating points, network configuration, type of used medium, communication protocol, or application using network resources [64]. Subsequent sections elaborate on some of the delineated metrics further.

Several approaches have been developed to evaluate overall network performance. Two generally accepted and prevalent approaches are the quality of service (QoS) and the quality of experience (QoE). While the former is rather a set of individual measurable parameters, such as those we have outlined in earlier sections, which the network equipment monitors and manages [65], the latter is a measurement of how the service is experienced by an end-user [66]. However, some applications may not be sensitive to QoS and QoE. As in control of industrial processes, telemetry is the opposite—an example of a service requiring real-time data streaming performance with an additional characteristic of zero tolerance for informational error and loss. Therefore, QoS requirements are defined tightly. In addition, QoS calculation and enforcement may be difficult to perform in some time-critical applications as its components may be handled by different underlying infrastructures [67]. On the other hand, QoE as a holistic approach focuses on the entirety of the service from the perspective of a user [68]. Even though QoE has been initiated as a subjective metric based on human ratings, there are numerous procedures to gather objective, algorithmic evaluation [69].

4. Offloading Mechanism Design and Implementation

The presented section is aimed at the implementation of the system for offloading strategy utilization as well as on the design of the strategy itself. As a result, the following sections are concerned with designing and deploying the system's components vital for the strategy deployment. Figure 7 displays the flow of the process in the system. Figure 8 portrays a high-level presentation of the proposed system's architecture with all of its primary components. Individual parts can be categorized into two primary groups, that is local and remote. The remote group is represented solely by the cloud computing environment and its mechanisms. In contrast, the local group contains both the UAV and the edge server. The proposed separation implies that while the cloud environment will be reachable via the public internet connection, both UAV and the edge server will be interconnected through the wireless local area network.

This separation follows principles described in earlier sections and sustains reliable connectivity between the edge and the UAV. The connection reliability and speediness are crucial due to the deployment of core modules on the edge server. This separation follows principles described in earlier sections and sustains reliable connectivity between the edge and the UAV. The connection reliability and speediness is crucial due to the deployment of core modules on the edge server of which the most important to ensure the basic functionality is the application server the UAV will communicate with in order to create requests for path plan generation and the local instance of the path planning module utilized for path plan generation. In addition, the edge server is responsible for the operation of components required for the offloading strategy realization. Those are components created to monitor and evaluate parameters of the network connection between the edge and the cloud as well as for monitoring of the problem parameters, that is, map occupancy ratio and predicted depth of the solution. Lastly, the classification model for appropriate action determination, whether the problem should be offloaded to the remote cloud environment or solved locally, is deployed in the edge. In contrast, with the edge complexity, the remote part is equipped solely with the path planning module instance.

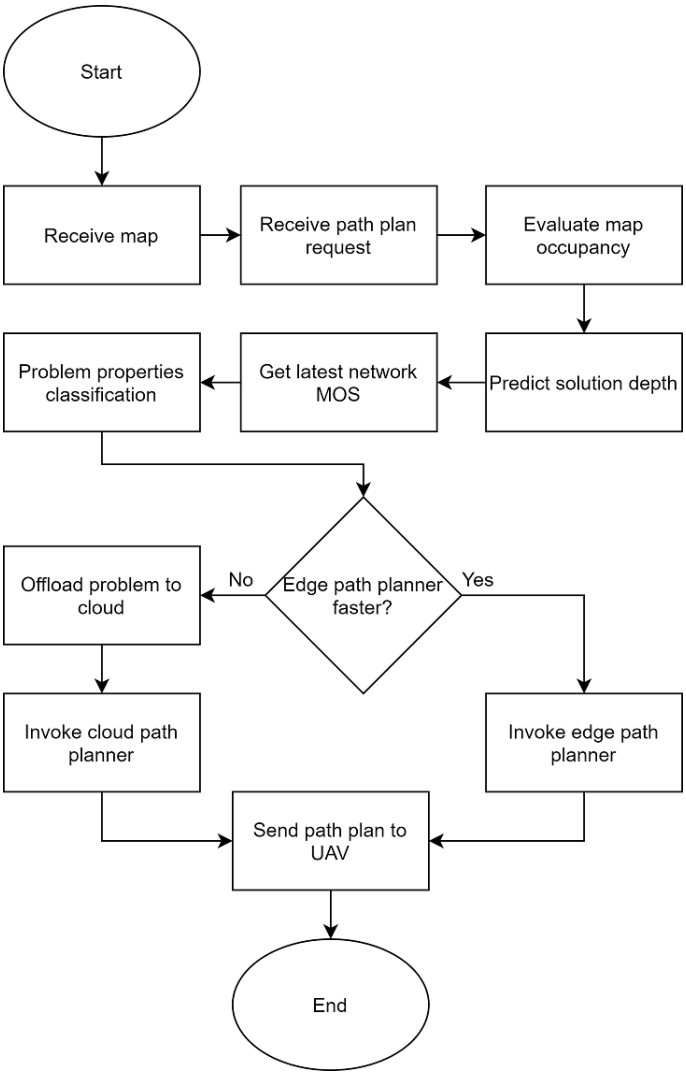

Figure 7. Flowchart of the offloading process.

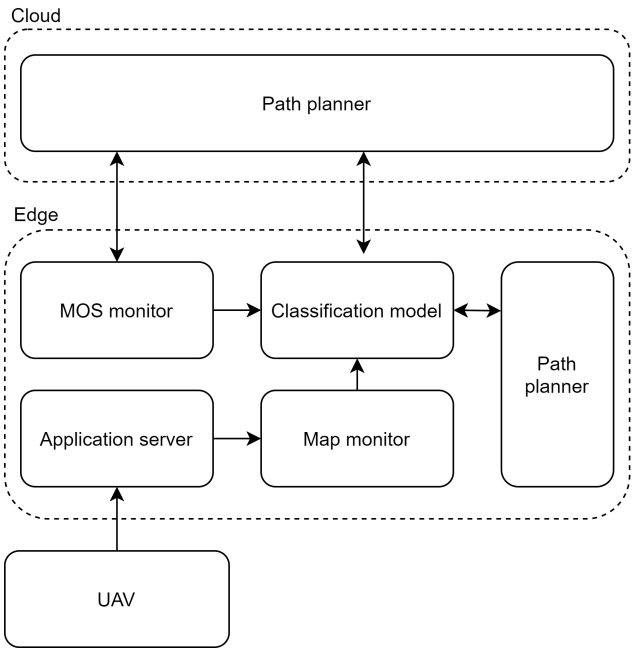

Figure 8. Architecture of the designed system.

4.1. Map Monitor

The map monitor as a component provides essential data for the classification system. The primary function of this monitor is the estimation of map occupancy rate, that is, the percentage of nodes in the map that are considered not transferable. That is achieved by iterative browsing throughout the nodes registered in the map and evaluating their state. This information is necessary for the other function, which is a prediction of the path length. Section 3.1 presented the relation of time complexity to the solution depth. In addition, path length in relation to map occupancy was investigated. Therefore, we regard solution depth as one of the viable parameters to automate the decision to offload a task from the edge to the cloud. However, a significant disadvantage of such a parameter is that the solution depth is known only after the algorithm has terminated. In order to address this drawback, we aimed at the construction of a model capable of predicting the depth of the solution prior to the algorithm execution being initiated.

Figure 9 presents the structure of the map monitor module. As illustrated, when a UAV issues a request to find an appropriate path plan, a map is sent to the application server. Consequently, the occupancy counter estimates the occupancy rate in the received map and determines the map size and outputs those data towards the linear regression model. Concerning the data presented in Figure 3, the model is built upon the least squares method:

$$y = \alpha + \beta x, \tag{3}$$

in which the estimated value $\hat{\beta}$ is calculated as:

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{4}$$

and consqeuently, the estimated $\hat{\alpha}$:

$$\hat{\alpha} = \bar{y} - (\hat{\beta}\bar{x}). \tag{5}$$

Therefore, after calculating respective values, this model can estimate multiples of the best-case solution, that is, a straight line, between two nodes by having information about the occupancy rate in the map. Hence, the system can predict the final solution length by trivial multiplication of the best-case solution depth.

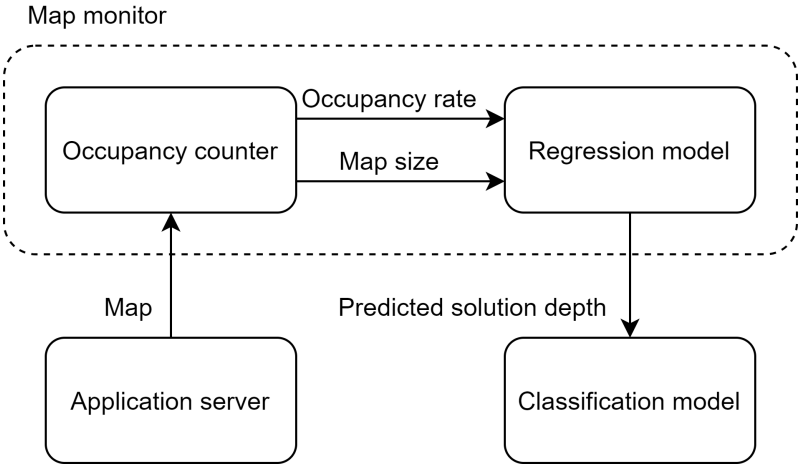

Figure 9. Detailed architecture of the map monitoring component.

4.2. MOS Monitor

The presented component is responsible for the monitoring and evaluation of network performance. Combined with the map monitor, it fulfills the role of providing essential information about the problem properties to the offloading strategy. The component

structure is presented in Figure 10. It employs three separate elements to measure metrics, namely RTT, transfer speed and packet drop rate, following definitions in Section 3.2. Lastly, the evaluation model calculates the network performance using the mean opinion score (MOS) model to provide a network rating for the classification model.

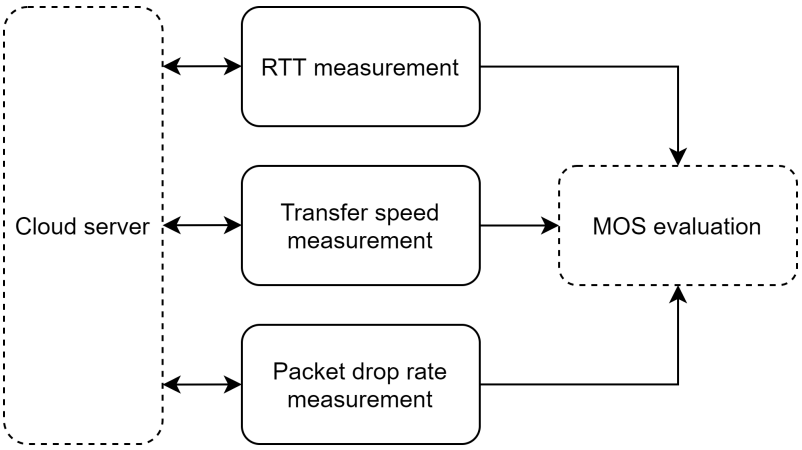

Figure 10. Detailed architecture of the MOS monitor component.

The mean opinion score is a widely used technique for the network performance evaluation to represent the system's comprehensive quality. Despite its common utilization of said model in assessing video or audio streams, there is no limitation to those services. The evaluation is based on the singular metrics performance categorization into five classes (5—excellent, 1—bad). According to the performance requirements for telemetry applications presented in Section 3.2, we split values for each metric in a manner delineated in Table 2.

Table 2. Definition of rating classes for needs of the MOS monitor.

Class	Value	RTT (ms)	Packet Drop (%)	Transfer Speed (Mbps)
Excellent	5	(0; 62)	0	(50; inf)
Good	4	(63; 125)	-	(50; 37.5)
Fair	3	(126; 187)	-	(37.5, 25)
Poor	2	(188; 250)	-	(25; 12.5)
Bad	1	(250; inf)	[1; 100]	(12.5; 0.002)

As the network quality evaluation of the system is based on the combination of outlined approaches (see Section 3.2), the system performs continual measurement of three metrics. The continual measurement is employed to avoid the offloading decision withheld due to the time required to assess the performance. However, continual measurement may observe the variance in the acquired results. Therefore, we utilize an exponential smoothing for each metric calculated as follows:

$$s_t = \alpha x_t + (1 - \alpha) s_{t-1} = s_{t-1} + \alpha(x_t - s_{t-1}). \tag{6}$$

The smoothing factor α is calculated as:

$$\alpha = 1 - e^{-\Delta T/\tau}; \tau = -\frac{\Delta T}{\ln(1-\alpha)}. \tag{7}$$

After smoothing, acquired values are tested whether they belong to a respective interval to assign an appropriate class. Subsequently, the mean opinion score can then be calculated as a simple arithmetic mean:

$$S = \frac{\sum_{n=1}^{N} V_n}{N}, \tag{8}$$

where V_n refers to the value of one of the rating classes described in Table 2, and the resulting scores will facilitate holistic network quality evaluation for the machine learning algorithms implemented in later sections.

4.3. Classification Model

The present section aims to utilize machine learning models for automated task offloading in edge and cloud-enabled systems. At first, a method of data collection employed for the model training is elaborated. Subsequently, we define two machine learning approaches, namely the k-nearest neighbors and the logistic regression model. The two supervised learning models are chosen to compare various methods in the discussed classification task. In addition, the choice of the classification algorithm is based on the premise that for the objective of determining whether a path planning task will be solved in a more time-effective manner in the cloud environment or the edge does not necessarily require attempts at the prediction of the exact algorithm runtime but rather a dichotomous classification, meaning the more edge-effective class and the more cloud-effective class should suffice.

4.3.1. Data Collection

To maintain a diminutive time of computation for not mitigating the potential benefits of the offloading mechanism, this system will utilize an offline learning method to train models. That implies the necessity to secure data collection prior to the training. Hence, a measurement to acquire a dataset viable for the training of both models was set up. Considering that the proposed algorithms shall realize the offloading strategy based on the properties of the map and the network connection between the edge and the cloud, components to evaluate problem parameters had to be involved alongside other components in the system.

Therefore, the dataset collection was carried out by iteratively issuing the simulated UAV requests towards the edge's application server. Those requests contained randomly generated maps and the required start and endpoints for the path plan. Subsequently, the map monitor had to estimate map occupancy and size and use the regression model for solution depth prediction. Concurrently, the MOS monitor measured and evaluated the properties of the network. Because there could not be any classification model included at this point, every request was routed towards the edge environment and later towards the cloud environment, where path planning modules were eventually invoked.

The dataset illustrated in Figure 11 depicts measurements labeled by the respective class according to the environment, where the solution was provided quicker. At this stage, the response from either environment was awaited, and the earlier one was recorded. In contrast with measurements presented in previous sections, results of this data collection do not reflect only the path planning algorithm runtime, but rather the time response of both systems with underlying configuration and software equipment with a specific implementation, which will likely affect the performance. Therefore, it is imperative to note that results may vary by the technology employed even when the implementation itself remains unaltered, which may signify the requirement to pursue new measurements for each deployment. Additionally, like previous experiments, the software artificially altered the network performance parameters considering there was no practical possibility of capturing variant network states in a timely manner.

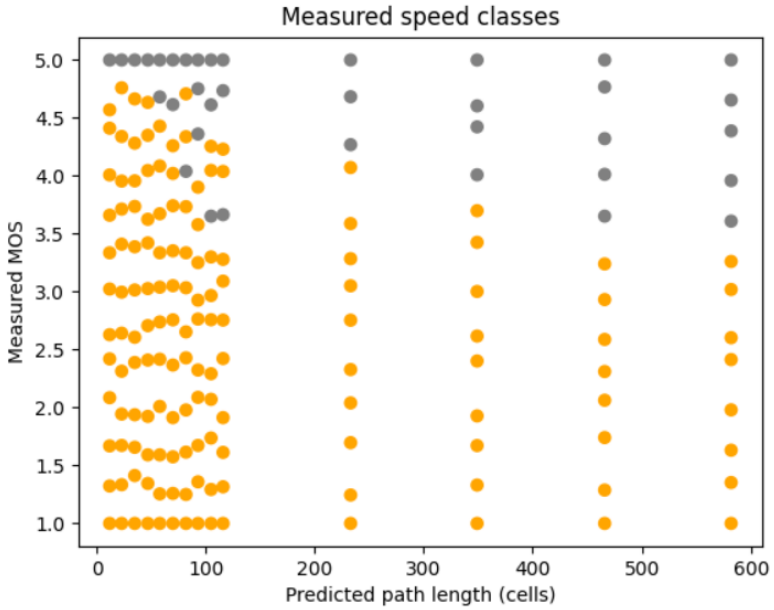

Figure 11. Collected dataset. Orange signifies edge was faster, grey signifies cloud was faster.

4.3.2. Classification Algorithms

As denoted earlier in this section, we have implemented two supervised machine learning algorithms for offloading strategy realization. Following the need for binary classification, we have established a set of labels:

$$y = \{0, 1\}, \tag{9}$$

where $y = 0$ denotes the computation shall be executed on edge, whereas $y = 1$ means offloading to the cloud.

The first implemented algorithm was the k-nearest neighbors built on the notion that the closest patterns to a concrete pattern for which a label is investigated provide valuable information [70]. Hence, we employed the p-norm as a similarity measure:

$$\left\| x^{'} - x_j \right\|^p = \left(\sum_{i=1}^{q} \left| (x_i^{'}) - (x_i)_j \right|^p \right)^{\frac{1}{p}}, \tag{10}$$

corresponding to the Euclidean distance in the case when $p = 2$. Accordingly, the classifier can be defined as:

$$f_c = \begin{Bmatrix} 1; & \sum_{i \epsilon N_K(\mathbf{x}^{'})} y_i \geq 0.5 \\ 0; & \sum_{i \epsilon N_K(\mathbf{x}^{'})} y_i < 0.5 \end{Bmatrix}, \tag{11}$$

where K indicates neighborhood size and $N_K(x^{'})$ is the set of indices of k-nearest neighbors [71]. By way of K defining the locality of the classifier, where lower values result in smaller neighborhoods. In order to determine the optimal K value, an experimental evaluation shall be performed to examine the best model accuracy.

The second algorithm tested for the designed solution was the logistic regression classifier estimating the probability of an input belonging to a class. Hence, the output of this classifier can be either 1, signifying a specific sample belongs to a class or 0 when the sample does not belong to a class. Therefore, the primary goal is the probability estimation $P(y = 1|x^{'})$. This task is elucidated by learning a vector of weights and a bias term from the training sample [72]. A weight's association with an input feature represents its importance for the classification decision. Additionally, the bias term is a real number added to the

weighted inputs. In order to make a decision after learning, the model calculates a weighted sum by the formula:

$$z = (\sum_{i=1}^{n} w_i x_i) + b. \tag{12}$$

The w_i represents individual weights and b is the bias. Because this sum is not guaranteed to result in the interval $[0, 1]$, it is subsequently passed to the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{13}$$

Next, we can define probabilities as:

$$\begin{array}{c} P(y = 1) = \sigma z, \\ P(y = 0) = 1 - \sigma z. \end{array} \tag{14}$$

Lastly, we can define the classifier function:

$$f_c = \begin{Bmatrix} 1; & if P(y = 1|x') > 0.5 \\ 0; & otherwise \end{Bmatrix}. \tag{15}$$

5. Experiments

The presented section provides insight into the performance of the system and its components. The experiments evaluate the monitors and the classification model, with attention given primarily to the prediction and classification. Therefore, a linear regression model is trained and tested for solution depth prediction accuracy as well as both the k-nearest neighbors, and the linear regression models are trained, tuned and evaluated. Lastly, tests to assess the time complexity of the solution were performed.

5.1. Solution Depth Prediction

In Section 4.1, we presented a linear regression model trained by the least-squares method for solution depth prediction based on the map occupancy. The model was trained on the data gathered and presented in Section 3.1, where the dependence of the path length on the map occupancy rate is graphed in Figure 3. Therefore, the model was trained with data acquired from randomly generated square-shaped maps.

The training yielded coefficients $\alpha = 0.007$ and $\beta = 0.9893$. Therefore, the model can be defined as follows:

$$y = 0.007x + 0.9893. \tag{16}$$

Resulting regression is graphed in Figure 12. For the evaluation of the model, we have calculated a set of metrics presented in Table 3.

Table 3. Evaluation metrics of the linear regression model.

Mean Absolute Error	Mean Squared Error	Coefficient of Determination
0.0065	6.7346	0.98644

From the observation of the three metrics presented and from the graphed model, it can be concluded that the model fits the data well and has a high prediction accuracy. Hence, the model is fit to provide predicted data to the classification model for the offloading strategy realization.

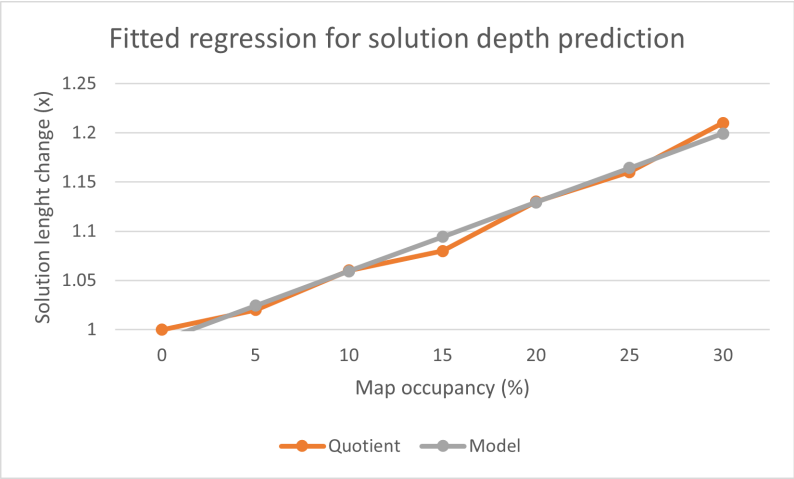

Figure 12. Trained linear regression model.

5.2. Classification Model Training

The essential component of the designed system is undoubtedly the classification model as a means to make offloading decisions. As it was stated in Section 4.3, we tested two supervised learning models for performance comparison. Both models were trained on the dataset containing 182 entries, as displayed in Section 4.3.1.

The first trained model is the k-nearest neighbour. In order to estimate the best value of K, we have performed a series of experiments incrementally, increasing the value and comparing the model accuracy. The best results on the available dataset were achieved when $K = 3$ resulted in 86% accuracy. In contrast, K values of 4 and greater resulted in the worst observed model accuracy of 81%. Therefore, any further implementation and evaluation continued with $K = 3$. Further insight into the model performance is depicted in Figure 13.

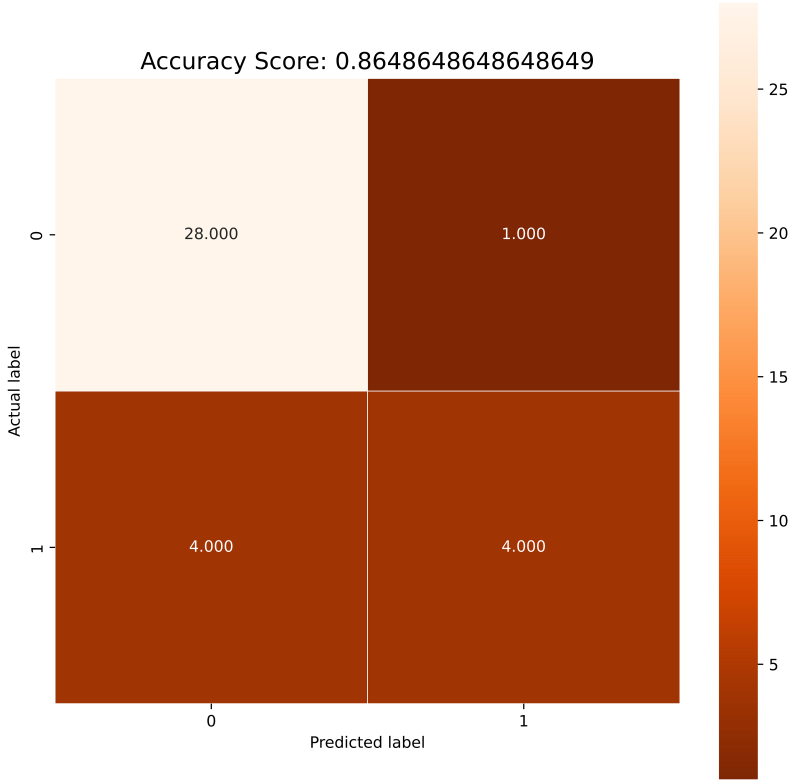

Figure 13. K-NN classifier confusion matrix.

The second trained model was the linear regression classifier. The training dataset remained identical to the one used for the K-NN training. The resulting accuracy of the model was measured at 94%. Further insight into the model's accuracy is illustrated in Figure 14, where the confusion matrix is presented. The achieved accuracy is higher with the logistic regression classifier in comparison with the best results of the previously presented K-NN model. Results of the KNN classifier performance with differing k values are presented in Figure 15.

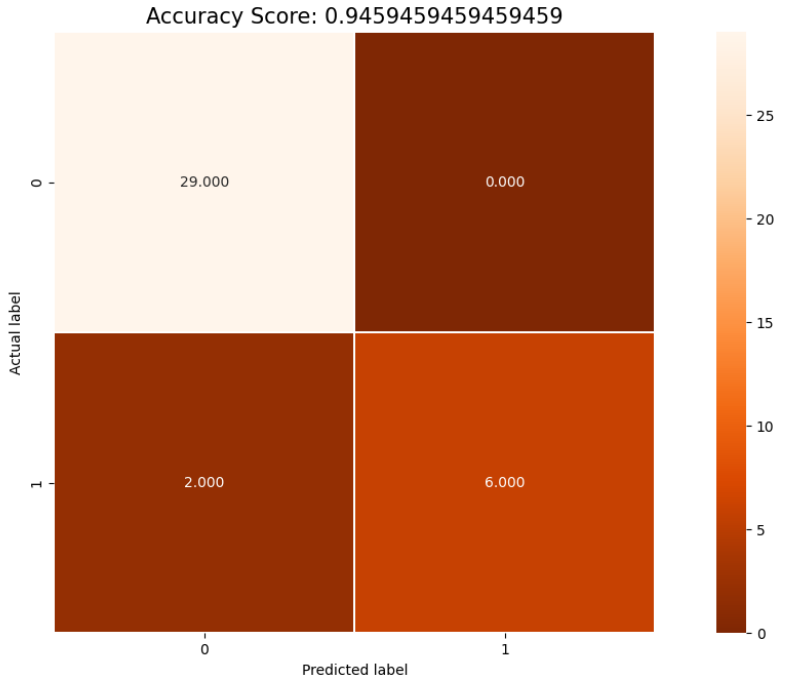

Figure 14. Linear regression confusion matrix.

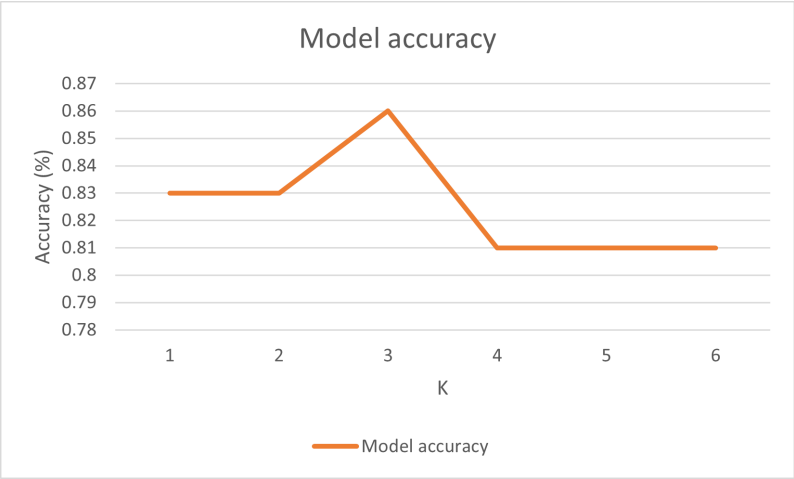

Figure 15. Accuracy of K-NN classifier with different K values.

5.3. System's Time Performance

Another evaluated aspect of the designed system is its time performance. In detail, we evaluate the time the classification models are required to learn on the dataset to determine and examine the time the system is required to make an offloading decision.

Following the prior statement, we have examined the learning times of the models with datasets of various sizes. The datasets contained, similarly to prior experiments,

entries with predicted solution depth, measured mean opinion score and information about the environment, in which the problem was solved more expeditiously. Measured learning times for both models are depicted in Figure 16.

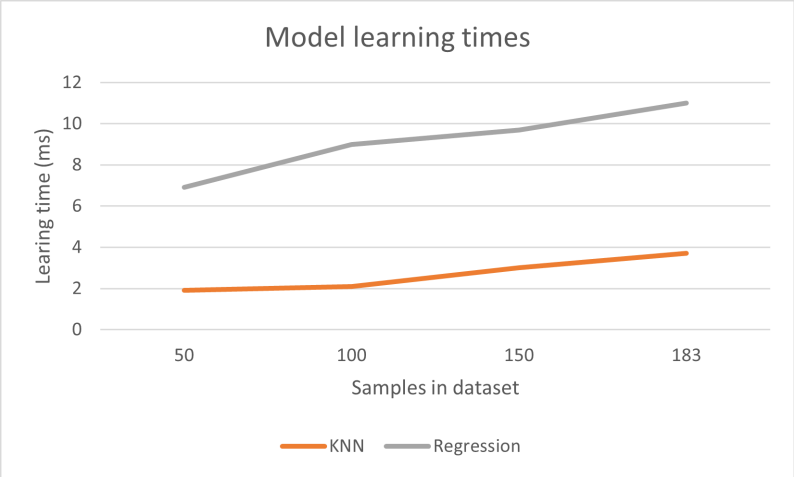

Figure 16. Learning times of classification models.

It is visible that the logistic regression model requires a significantly longer time to learn. However, it achieves greater accuracy in classification. In addition, the curve for the logistic regression shows a steeper increase compared to the K-NN model. Hence, we can conclude that offline training may significantly improve the time required to make an offloading decision compared to online and especially batch training. On the contrary, online or batch training would be more suitable in combination with the K-NN model.

Subsequently, we have performed a set of measurements to estimate the time the system required to decide whether the problem should be offloaded to the cloud environment or solved on edge to assess whether the system utilization shall not suppress potential benefits of effective problem offloading. To assess the time, we have employed the following formula:

$$TTC = Retrieval_{MOS} + Retrieval_{Occupancy} + Prediction_{Depth} + Classification_{Offloading}. \quad (17)$$

In the formula, the TTC is the time to classification, $Retrieval_{MOS}$ is the time needed to retrieve the latest MOS score, $Retrieval_{Occupancy}$ represents the time the map monitor requires to estimate map occupancy rate, $Prediction_{Depth}$ is the time required to predict solution depth, and finally, $Classification_{Offloading}$ is the time either model requires to classify input data. Figure 17 presents measurement results. In accordance with displayed times, the recorded difference between the two classification models was not significant. Subsequently, the measured values did not show significant growth when the side of the map was not greater than 100 cells. With larger maps, however, the time to classify grew steeply. This phenomenon was caused by the operation of the map monitor and its evaluation of occupancy rate. Although the time increases steeply with larger maps, it is important to note that it is where the system shows the most benefit because with maps having smaller sides than 100 cells, the time only to make a decision is almost equal to the time to solve the path planning problem in either environment (see Figure 1).

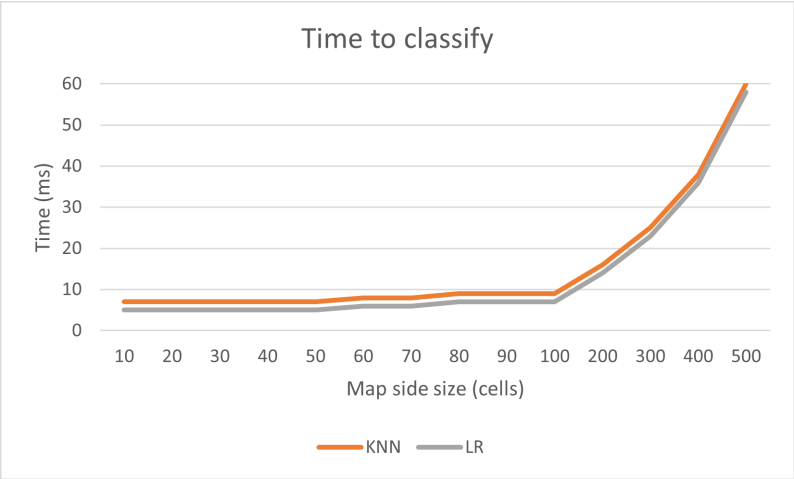

Figure 17. Time required to make an offloading decision.

5.4. Overall Solution Time

In the last evaluation, we have measured differences in the time the system requires to produce a solution, that is, to create a path plan from the moment a request for a path is received. This period, therefore, includes the time required to make an offloading decision, time to potentially offload the problem, as well as path plan generation itself. In order to perform the measurement, we have set up four scenarios:

- Exclusive edge utilization,
- Exclusive cloud utilization,
- Offloading mechanism with KNN classifier,
- Offloading mechanism with an LR classifier.

In addition, maps with side sizes ranging from 10 to 500 cells were generated with random occupancy rates within the constraints of the earlier described map generator. Similarly, MOS values were altered artificially and randomly; however, those were in ranges that were still allowed to reach the cloud. Results are presented in Figure 18.

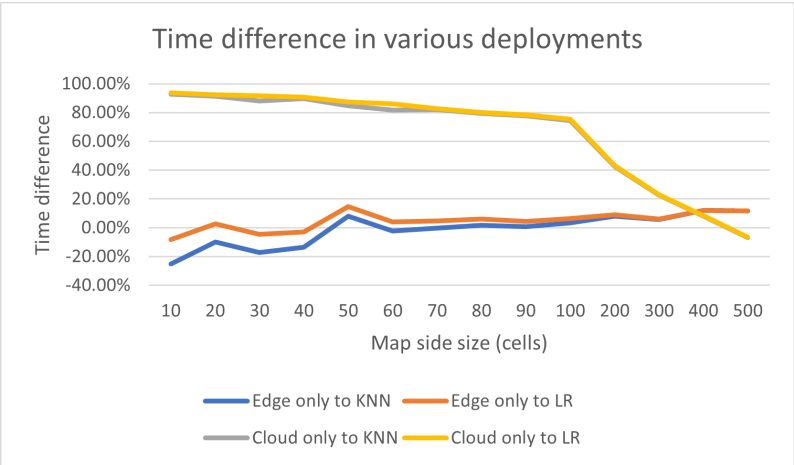

Figure 18. Time differences in various deployment scenarios.

It can be stated that when the map side size did not exceed 50 cells, it was up to 25.22% faster at 10 cells when the computation was performed only at the edge, and the offloading mechanism was not employed. Consequently, with large maps of a side size of at least 400 cells, the scenario when the only cloud was employed showed a negative time difference up to 6.72%; therefore, employment of only the cloud would yield a path plan faster. In contrast, utilization of the offloading mechanism has shown up to 93.74% saved

time with small maps compared to cloud-only deployment and 11.76% in comparison with edge-only employment.

Therefore, differences in solution time are smaller with large maps compared to differences in path plan calculation due to the response time from the cloud.

6. Conclusions

This work has presented a system for automated computation offloading for path planning problems in edge-enabled cloud arrangements to expedite pathfinding for unmanned aerial vehicles. The primary objective is based upon the prediction of time a path planning algorithm will need to solve in either a cloud or edge computing environment. In contrast with some existing deployments, this automated offloading does not try to predict the exact run time for either environment but instead focuses on determining which of the two will enable the more expeditious solution to a planning problem. In order to do so, the system is equipped with monitors responsible for assessing both qualities of the network, critical due to the necessity to transfer data between the edge and cloud and for monitoring the map on which a path has to be found. That is achieved by evaluating map occupancy and size, allowing a regression model to predict the length of a potential path. Information provided by those monitors enables supervised learning classification models to estimate a potentially faster destination for problem-solving while considering computational power and network quality.

The experimental evaluation has demonstrated that such an approach to automated offloading strategies is feasible and may be beneficial in reducing computation time compared to exclusive utilization of the cloud or the edge. However, results have shown that extremal map sizes may have an opposite effect. A possible solution to this problem may be setting the threshold level, which would turn off the offloading mechanism and automatically forward the problem to the appropriate environment. In addition, the experimentation has demonstrated that times required to perform an offloading decision are only up to 60ms with the largest maps, which conclude that the mechanism does not impose great computational effort on the engines.

However, the system was able to make decisions; we perceive several areas for possible improvements. The main limitation of the implemented arrangement is the necessity for offline learning, which requires data collection related to the problem prior to the system employment. That may be a significant limiting factor since recording various network quality states might be time-consuming outside of laboratory conditions. Additionally, a possibility to utilize unsupervised machine learning algorithms should be examined to evade the necessity for data labeling, or an automated labeling system should be considered [73].

An objective comparison of the proposed system to other approaches may be problematic due to the usage of different technologies and offloaded problems. However, in contrast to some other solutions that concern path planning offloading, the proposed system considers computation times, as opposed to energy consumption [43].

However, systems that offload path planning problems have demonstrated up to 80% time saving with offloading decisions performed by using a resource-block based model [74]. Additionally, the proposed solution shows shorter learning times as well as shorter time to make a decision [75]. For the purposes of such comparison, it is essential to note that the resulting time conservation may be skewed by the underlying technology used in the implementation.

Subsequently, the system may be expanded to serve multiple UAVs concurrently, which shall require consideration of additional parameters such as the construction of mechanisms to monitor queues in both environments.

Lastly, any expansion of this approach should be preceded by the system's optimization, that is, the enhancement of its components, primarily of the map monitor, which is rendered to be the primary speed decelerator by utilization of speedier algorithms and by enhancement of the system as a whole. This can be achieved by exchange of the technol-

ogy on which the arrangement is deployed and by utilization of concurrent computation paradigms, e.g., using evolutionary computing [76].

Author Contributions: Conceptualization and methodology, D.H.; formal analysis and supervision, J.V.; writing—original draft preparation, D.H., J.V. and A.B.; funding acquisition and project administration, I.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This publication is the result of the APVV grant ENISaC—edge-enabled intelligent sensing and computing (APVV-20-0247).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

KNN	k-nearest neighbors
MOS	mean opinion score
NIST	National Institute of Standards and Technology
QoE	quality of experience
QoS	quality of service
RTT	round trip time
UAV	unmanned aerial vehicle

References

1. Gilmore, C.K.; Chaykowsky, M.; Thomas, B. *Autonomous Unmanned Aerial Vehicles for Blood Delivery: A UAV Fleet Design Tool and Case Study*; Technical Report; Rand Arroyo Center: Santa Monica, CA, USA, 2019.
2. Lygouras, E.; Santavas, N.; Taitzoglou, A.; Tarchanidis, K.; Mitropoulos, A.; Gasteratos, A. Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations. *Sensors* **2019**, *19*, 3542. [CrossRef] [PubMed]
3. Poljak, M.; Šterbenc, A. Use of drones in clinical microbiology and infectious diseases: Current status, challenges and barriers. *Clin. Microbiol. Infect.* **2020**, *26*, 425–430. [CrossRef]
4. Azadeh, K.; De Koster, R.; Roy, D. Robotized warehouse systems: Developments and research opportunities. In *ERIM Report Series Research in Management Erasmus Research Institute of Management*; Rotterdam School of Management (RSM), Erasmus University: Rotterdam, The Netherlands, 2017.
5. Saha, O.; Dasgupta, P. A comprehensive survey of recent trends in cloud robotics architectures and applications. *Robotics* **2018**, *7*, 47. [CrossRef]
6. Abouzahir, M.; Elouardi, A.; Latif, R.; Bouaziz, S.; Tajer, A. Embedding SLAM algorithms: Has it come of age? *Robot. Auton. Syst.* **2018**, *100*, 14–26. [CrossRef]
7. Hamid, U.Z.A.; Saito, Y.; Zamzuri, H.; Rahman, M.A.A.; Raksincharoensak, P. A review on threat assessment, path planning and path tracking strategies for collision avoidance systems of autonomous vehicles. *Int. J. Veh. Auton. Syst.* **2018**, *14*, 134–169. [CrossRef]
8. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [CrossRef]
9. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2011.
10. Mohanarajah, G.; Hunziker, D.; D'Andrea, R.; Waibel, M. Rapyuta: A cloud robotics platform. *IEEE Trans. Autom. Sci. Eng.* **2014**, *12*, 481–493. [CrossRef]
11. Riazuelo, L.; Civera, J.; Montiel, J.M. C2tam: A cloud framework for cooperative tracking and mapping. *Robot. Auton. Syst.* **2014**, *62*, 401–413. [CrossRef]
12. Hong, C.; Shi, D. A cloud-based control system architecture for multi-UAV. In Proceedings of the 3rd International Conference on Robotics, Control and Automation, Chengdu, China, 11–13 August 2018; pp. 25–30.
13. Dolui, K.; Datta, S.K. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In Proceedings of the 2017 Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6. [CrossRef]

14. Messous, M.A.; Hellwagner, H.; Senouci, S.M.; Emini, D.; Schnieders, D. Edge computing for visual navigation and mapping in a UAV network. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.
15. Pozna, C.; Precup, R.E.; Foeldesi, P. A novel pose estimation algorithm for robotic navigation. *Robot. Auton. Syst.* **2015**, *63*, 10–21. [CrossRef]
16. Hayat, S.; Jung, R.; Hellwagner, H.; Bettstetter, C.; Emini, D.; Schnieders, D. Edge computing in 5G for drone navigation: What to offload? *IEEE Robot. Autom. Lett.* **2021**, *6*, 2571–2578. [CrossRef]
17. Štefanič, P.; Cigale, M.; Jones, A.C.; Knight, L.; Taylor, I.; Istrate, C.; Suciu, G.; Ulisses, A.; Stankovski, V.; Taherizadeh, S.; et al. SWITCH workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications. *Future Gener. Comput. Syst.* **2019**, *99*, 197–212. [CrossRef]
18. Angelov, P.; Škrjanc, I.; Blažič, S. Robust evolving cloud-based controller for a hydraulic plant. In Proceedings of the 2013 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS), Singapore, 16–19 April 2013; pp. 1–8. [CrossRef]
19. Wan, J.; Tang, S.; Yan, H.; Li, D.; Wang, S.; Vasilakos, A.V. Cloud robotics: Current status and open issues. *IEEE Access* **2016**, *4*, 2797–2807. [CrossRef]
20. Chen, W.; Yaguchi, Y.; Naruse, K.; Watanobe, Y.; Nakamura, K. QoS-aware robotic streaming workflow allocation in cloud robotics systems. *IEEE Trans. Serv. Comput.* **2018**, *14*, 544–558. [CrossRef]
21. Li, S.; Zheng, Z.; Chen, W.; Zheng, Z.; Wang, J. Latency-aware task assignment and scheduling in collaborative cloud robotic systems. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 65–72.
22. Chifamba, S. A Study on Cloud Robotics: Ad-hoc cloud (Cloud Seeding). *Int. J. Innov. Res. Comput. Commun. Eng.* **2015**, *3*, 3219–3226.
23. McGilvary, G.A.; Barker, A.; Atkinson, M. Ad Hoc Cloud Computing. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 1063–1068. [CrossRef]
24. Beigi, N.K.; Partov, B.; Farokhi, S. Real-time cloud robotics in practical smart city applications. In Proceedings of the 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, 8–13 October 2017; pp. 1–5. [CrossRef]
25. Antevski, K.; Groshev, M.; Cominardi, L.; Bernardos, C.; Mourad, A.; Gazda, R. Enhancing edge robotics through the use of context information. In Proceedings of the Workshop on Experimentation and Measurements in 5G, Heraklion, Greece, 4 December 2018; pp. 7–12.
26. Huaimin, W.; Bo, D.; Xu, J. Cloud Robotics: A Distributed Computing View. In *Symposium on Real-Time and Hybrid Systems: Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*; Jones, C., Wang, J., Zhan, N., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 231–245. [CrossRef]
27. Sarker, V.K.; Queralta, J.P.; Gia, T.N.; Tenhunen, H.; Westerlund, T. Offloading slam for indoor mobile robots with edge-fog-cloud computing. In Proceedings of the 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), Dhaka, Bangladesh, 3–5 May 2019; pp. 1–6.
28. Wang, T.; Zhao, D.; Cai, S.; Jia, W.; Liu, A. Bidirectional Prediction-Based Underwater Data Collection Protocol for End-Edge-Cloud Orchestrated System. *IEEE Trans. Ind. Inform.* **2020**, *16*, 4791–4799. [CrossRef]
29. Aissioui, A.; Ksentini, A.; Gueroui, A.M.; Taleb, T. On Enabling 5G Automotive Systems Using Follow Me Edge-Cloud Concept. *IEEE Trans. Veh. Technol.* **2018**, *67*, 5302–5316. [CrossRef]
30. Wang, P.; Yang, L.T.; Li, J. An Edge Cloud-Assisted CPSS Framework for Smart City. *IEEE Cloud Comput.* **2018**, *5*, 37–46. [CrossRef]
31. Shi, S.; Gupta, V.; Hwang, M.; Jana, R. Mobile VR on edge cloud: A latency-driven design. In Proceedings of the 10th ACM Multimedia Systems Conference, Amherst, MA, USA, 18–21 June 2019; pp. 222–231.
32. Wang, T.; Liang, Y.; Zhang, Y.; Zheng, X.; Arif, M.; Wang, J.; Jin, Q. An Intelligent Dynamic Offloading From Cloud to Edge for Smart IoT Systems With Big Data. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 2598–2607. [CrossRef]
33. Chen, Y.; Zhang, N.; Zhang, Y.; Chen, X.; Wu, W.; Shen, X.S. Energy efficient dynamic offloading in mobile edge computing for Internet of Things. *IEEE Trans. Cloud Comput.* **2019**, *9*, 1050–1060. [CrossRef]
34. Zhang, Y.; He, J.; Guo, S. Energy-efficient dynamic task offloading for energy harvesting mobile cloud computing. In Proceedings of the 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), Chongqing, China, 11–14 October 2018; pp. 1–4.
35. Li, M.; Wu, Q.; Zhu, J.; Zheng, R.; Zhang, M. A computing offloading game for mobile devices and edge cloud servers. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 2179316. [CrossRef]
36. Yousafzai, A.; Yaqoob, I.; Imran, M.; Gani, A.; Md Noor, R. Process Migration-Based Computational Offloading Framework for IoT-Supported Mobile Edge/Cloud Computing. *IEEE Internet Things J.* **2020**, *7*, 4171–4182. [CrossRef]
37. Wu, H.; Wolter, K.; Jiao, P.; Deng, Y.; Zhao, Y.; Xu, M. EEDTO: An Energy-Efficient Dynamic Task Offloading Algorithm for Blockchain-Enabled IoT-Edge-Cloud Orchestrated Computing. *IEEE Internet Things J.* **2021**, *8*, 2163–2176. [CrossRef]
38. Peng, K.; Huang, H.; Wan, S.; Leung, V.C. End-edge-cloud collaborative computation offloading for multiple mobile users in heterogeneous edge-server environment. *Wirel. Netw.* **2020**, 1–12. [CrossRef]

39. Ning, Z.; Dong, P.; Wang, X.; Guo, L.; Rodrigues, J.J.P.C.; Kong, X.; Huang, J.; Kwok, R.Y.K. Deep Reinforcement Learning for Intelligent Internet of Vehicles: An Energy-Efficient Computational Offloading Scheme. *IEEE Trans. Cogn. Commun. Netw.* **2019**, *5*, 1060–1072. [CrossRef]
40. Qiu, C.; Wang, X.; Yao, H.; Du, J.; Yu, F.R.; Guo, S. Networking Integrated Cloud–Edge–End in IoT: A Blockchain-Assisted Collective *Q*-Learning Approach. *IEEE Internet Things J.* **2021**, *8*, 12694–12704. [CrossRef]
41. Wang, X.; Han, Y.; Leung, V.C.; Niyato, D.; Yan, X.; Chen, X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
42. Tuli, S.; Ilager, S.; Ramamohanarao, K.; Buyya, R. Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks. *IEEE Trans. Mob. Comput.* **2020**. [CrossRef]
43. Spatharakis, D.; Avgeris, M.; Athanasopoulos, N.; Dechouniotis, D.; Papavassiliou, S. A Switching Offloading Mechanism for Path Planning and Localization in Robotic Applications. In Proceedings of the 2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), Rhodes Island, Greece, 2–6 November 2020; pp. 77–84. [CrossRef]
44. Le, A.V.; Prabakaran, V.; Sivanantham, V.; Mohan, R.E. Modified a-star algorithm for efficient coverage path planning in tetris inspired self-reconfigurable robot with integrated laser sensor. *Sensors* **2018**, *18*, 2585. [CrossRef] [PubMed]
45. Yang, D.; Xu, B.; Rao, K.; Sheng, W. Passive infrared (PIR)-based indoor position tracking for smart homes using accessibility maps and a-star algorithm. *Sensors* **2018**, *18*, 332. [CrossRef] [PubMed]
46. Zheng, T.; Xu, Y.; Zheng, D. AGV path planning based on improved A-star algorithm. In Proceedings of the 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Chongqing, China, 11–13 October 2019; pp. 1534–1538.
47. Zhang, Y.; Li, L.l.; Lin, H.C.; Ma, Z.; Zhao, J. Development of path planning approach using improved A-star algorithm in AGV system. *J. Internet Technol.* **2019**, *20*, 915–924.
48. Tucnik, P.; Nachazel, T.; Cech, P.; Bures, V. Comparative analysis of selected path-planning approaches in large-scale multi-agent-based environments. *Expert Syst. Appl.* **2018**, *113*, 415–427. [CrossRef]
49. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
50. Liu, X.; Gong, D. A comparative study of A-star algorithms for search and rescue in perfect maze. In Proceedings of the 2011 International Conference on Electric Information and Control Engineering, Wuhan, China, 25–27 March 2011; pp. 24–27.
51. Rios, L.H.O.; Chaimowicz, L. A survey and classification of A* based best-first heuristic search algorithms. In *Brazilian Symposium on Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 253–262.
52. Sun, D.; Li, M. Evaluation function optimization of A-star algorithm in optimal path selection. *Rev. Tec. De La Fac. De Ing. Univ. Del Zulia* **2016**, *39*, 105–111.
53. Brewka, G. Artificial intelligence—A modern approach by Stuart Russell and Peter Norvig, Prentice Hall. Series in Artificial Intelligence, Englewood Cliffs, NJ. *Knowl. Eng. Rev.* **1996**, *11*, 78–79. [CrossRef]
54. Wang, J.; Pan, J.; Esposito, F.; Calyam, P.; Yang, Z.; Mohapatra, P. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–23. [CrossRef]
55. Shakarami, A.; Ghobaei-Arani, M.; Masdari, M.; Hosseinzadeh, M. A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective. *J. Grid Comput.* **2020**, *18*, 639–671. [CrossRef]
56. Tao, X.; Ota, K.; Dong, M.; Qi, H.; Li, K. Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing. *IEEE Wirel. Commun. Lett.* **2017**, *6*, 774–777. [CrossRef]
57. Jadeja, Y.; Modi, K. Cloud computing-concepts, architecture and challenges. In Proceedings of the 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Nagercoil, India, 21–22 March 2012; pp. 877–880.
58. Li, W.; Xia, Y.; Zhou, M.; Sun, X.; Zhu, Q. Fluctuation-Aware and Predictive Workflow Scheduling in Cost-Effective Infrastructure-as-a-Service Clouds. *IEEE Access* **2018**, *6*, 61488–61502. [CrossRef]
59. Wu, Q.; Zhang, R. Towards smart and reconfigurable environment: Intelligent reflecting surface aided wireless network. *IEEE Commun. Mag.* **2019**, *58*, 106–112. [CrossRef]
60. Chen, Y.; Farley, T.; Ye, N. QoS requirements of network applications on the Internet. *Inf. Knowl. Syst. Manag.* **2004**, *4*, 55–76.
61. Mirkovic, D.; Armitage, G.; Branch, P. A Survey of Round Trip Time Prediction Systems. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1758–1776. [CrossRef]
62. Liu, W.; Zhao, D.; Zhu, G. End-to-end delay and packet drop rate performance for a wireless sensor network with a cluster-tree topology. *Wirel. Commun. Mob. Comput.* **2014**, *14*, 729–744. [CrossRef]
63. Luz, T.C.; Margi, C.B.; Verdi, F.L. Network metrics detection to support internet of things application orchestration. *Open J. Internet Things (OJIOT)* **2021**, *7*, 93–103.
64. Abdulwahid, M.; Al-Hakeem, M.; Mosleh, M.; Abd-alhmeed, R. Investigation and optimization method for wireless AP deployment based indoor network. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2020; Volume 745, p. 012031.
65. Batarseh, F.A.; Latif, E.A. Assessing the quality of service using Big Data analytics: With application to healthcare. *Big Data Res.* **2016**, *4*, 13–24. [CrossRef]

66. Streijl, R.C.; Winkler, S.; Hands, D.S. Mean opinion score (MOS) revisited: Methods and applications, limitations and alternatives. *Multimed. Syst.* **2016**, *22*, 213–227. [CrossRef]
67. Koulouzis, S.; Martin, P.; Zhou, H.; Hu, Y.; Wang, J.; Carval, T.; Grenier, B.; Heikkinen, J.; De Laat, C.; Zhao, Z. Time-critical data management in clouds: Challenges and a Dynamic Real-Time Infrastructure Planner (DRIP) solution. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5269. [CrossRef]
68. Jain, R. Quality of experience. *IEEE Multimed.* **2004**, *11*, 95–96. [CrossRef]
69. Zhang, J.; Kuo, C.C.J. An objective quality of experience (QoE) assessment index for retargeted images. In Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 257–266.
70. Saadatfar, H.; Khosravi, S.; Joloudari, J.H.; Mosavi, A.; Shamshirband, S. A new K-nearest neighbors classifier for big data based on efficient data pruning. *Mathematics* **2020**, *8*, 286. [CrossRef]
71. Kramer, O. *Dimensionality Reduction with Unsupervised Nearest Neighbors*; Springer: Berlin/Heidelberg, Germany, 2013.
72. Zheng, X.; Rong, Y.; Liu, L.; Cheng, W. A More Accurate Estimation of Semiparametric Logistic Regression. *Mathematics* **2021**, *9*, 2376. [CrossRef]
73. Pozna, C.; Precup, R.E. Applications of Signatures to Expert Systems Modelling. *Acta Polytech. Hung.* **2014**, *11*, 21–39.
74. Li, X.; Dang, Y.; Aazam, M.; Peng, X.; Chen, T.; Chen, C. Energy-Efficient Computation Offloading in Vehicular Edge Cloud Computing. *IEEE Access* **2020**, *8*, 37632–37644. [CrossRef]
75. Xiao, S.; Wang, S.; Zhuang, J.; Wang, T.; Liu, J. Research on a task offloading strategy for the internet of vehicles based on reinforcement learning. *Sensors* **2021**, *21*, 6058. [CrossRef] [PubMed]
76. Johanyák, Z.C. A modified particle swarm optimization algorithm for the optimization of a fuzzy classification subsystem in a series hybrid electric vehicle. *Teh. Vjesn.* **2017**, *24*, 295–301.