Data-Driven Analytics Task Management Reasoning Mechanism in Edge Computing

Christos Anagnostopoulos 1,* , Tahani Aladwani 1, Ibrahim Alghamdi 2 and Konstantinos Kolomvatsos 3

1 School of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK; 2587711A@student.gla.ac.uk
2 Computer Science Department, Al-Baha University, Al Baha 65731, Saudi Arabia; ia.alghamdi@bu.edu.sa
3 Department of Informatics and Telecommunications, University of Thessaly, 382 21 Volos, Greece;
kostasks@uth.gr
* Correspondence: christos.anagnostopoulos@glasgow.ac.uk

Abstract: Internet of Things (IoT) applications have led to exploding contextual data for predictive analytics and exploration tasks. Consequently, computationally data-driven tasks at the network edge, such as machine learning models’ training and inference, have become more prevalent. Such tasks require data and resources to be executed at the network edge, while transferring data to Cloud servers negatively affects expected response times and quality of service (QoS). In this paper, we study certain computational offloading techniques in autonomous computing nodes (ANs) at the edge. ANs are distinguished by limited resources that are subject to a variety of constraints that can be violated when executing analytical tasks. In this context, we contribute a task-management mechanism based on approximate fuzzy inference over the popularity of tasks and the percentage of overlapping between the data required by a data-driven task and data available at each AN. Data-driven tasks’ popularity and data availability are fed into a novel two-stages Fuzzy Logic (FL) inference system that determines the probability of either executing tasks locally, offloading them to peer ANs or offloading to Cloud. We showcase that our mechanism efficiently derives such probability per each task, which consequently leads to efficient uncertainty management and optimal actions compared to benchmark models.

Keywords: edge computing; task offloading; data-driven analytics tasks; tasks popularity; fuzzy inference

1. Introduction

Autonomous driving, smart cities services, and augmented reality (AR) are just a few of new computational-intensive and data-driven applications over the IoT infrastructure [1]. Many of these applications are delay-sensitive and necessitate predictive, analytics and machine learning processes that are thought to be beyond the capability of end-user devices [2]. Cloud computing has been considered as the main solution to reduce the burden of data-driven tasks on edge devices. On the one hand, cloud computing is not the ideal option for delay-sensitive applications because analytic tasks cannot be completed in a real-time manner. This is owing to the fact that cloud data centers are typically located in places far away from real data sources. As a result, data processing in the cloud will eventually require increased communication activities via wide-area networks (WANs). This increases traffic in the network, the probability of tasks failures, and evidently results in relatively high response times [3,4]. Mobile Edge Computing (MEC), on the other hand, has already been adopted as a middle layer between the Cloud layer and the Sensing device layer, since various resources (e.g., computation and storage) could be utilized through LAN (Local Area Network) in MEC architecture. Using MEC resources to execute data-driven tasks (such as predictive modeling and analytic applications built for Unmanned Vehicles on mobile computing nodes) has brought many benefits for the end-users, such as reducing the pressure on the cloud, traffic bottleneck reduction, increase
bandwidth availability, and executing data-driven tasks in real-time [5–7]. Therefore, under MEC-Cloud computing paradigm, an entity called AN is introduced to provide both computation and transmission services in dynamic environments. However, in an MEC context, ANs have limited capacity and energy, making holistic task execution of all users’ requests by these nodes infeasible or very time consuming, especially when these tasks involve data processing and analytics (e.g., ML models training, outliers detection, data clustering, and classification tasks) [5,8]. Therefore, implementing a data-driven task management mechanism is critical for distributing tasks among ANs and the cloud according to specific criteria such as task urgency in terms of delay, task demand rate, task data accessibility, and the probability of re-using the same tasks in the future. All these factors could dramatically increase the utilization of MEC resources effectively [9], reduce the delivery delay, and improve QoS.

In this article, we propose a data-driven task-management mechanism based on the popularity of each task, which is derived from monitoring the most recent historical task demand rate within a sliding (time) window. According to task popularity, the mechanism can identify which tasks have been recently requested by many applications (end-users) and determine whether to locally execute or offload these tasks based on their statistical popularity. Notably, identifying low-popular tasks could help the mechanism avoid executing these tasks locally in ANs; instead, these tasks could be either sent to the cloud or to neighboring ANs (i.e., tasks have different popularity in different ANs), whereas highly popular tasks can be executed locally. However, most, not to mention all, analytic tasks require direct access to the distributed datasets stored locally on ANs. Hence, task offloading decisions should take into consideration the amount of data that need to be accessed by the tasks. In this context, our mechanism further introduces the data availability constraint in task-offloading decision making. The amount of data needed per task drives the decision making on whether to offload a task to neighboring ANs (peer ANs) based on their data availability or to the cloud.

What is noteworthy is that both of these factors (popularity of a task and amount of required data) significantly affect the offloading decision making for analytical tasks. However, in certain cases, these factors can possibly work in opposite directions, e.g., high task popularity and low data availability or vice versa. This requires the ability of the decision-making mechanism on an AN to swiftly balance between these factors by incorporating information from nearby ANs. This information includes the estimation of task popularity and the amount of data being available per task in an AN. In order to efficiently and effectively handle inherent uncertainties from these estimations (locally obtained from ANs), we propose the adoption of the Fuzzy Inference to represent decision-making rules based on the principles of Fuzzy Logic (FL). FL has been employed to determine the probability of offloading a task according to these factors. The main contribution of this work is summarized as follows:

- An architecture of collaboration between ANs and cloud computing is introduced. Furthermore, the design of ANs has been detailed.
- A novel offloading decision-making mechanism has been introduced based on tasks’ popularity, outliers, and data availability.
- A novel two-stages FL system has been developed to determine the probability of offloading for each task.
- A comprehensive comparative assessment of the proposed method against alternative mechanisms found in the literature showcase the applicability of our paradigm in edge computing environments.

The remainder of this paper is organized as follows. Section 2 elaborates on the related work in task offloading in EC environments, while Section 3 introduces an overview of the meaning of data-driven tasks, and the design of service architecture. Section 4 provides the problem formulation in our context and the task management factors. Section 5 explains task management reasoning and Section 6 reports on our experimental evolution. Section 7 concludes this paper with our future research agenda on this direction.
2. Related Work

2.1. Task Offloading

Due to resource, energy, and storage limitations in MEC, selecting the correct tasks management mechanism is a crucial issue, since MEC nodes deal with a huge volume of requests from end-users/applications [10]. Therefore, selecting the right tasks management mechanism can improve computing performance, execute the tasks in real-time, lower system costs, and maximize the use of available computing resources. With the deepening of research, several mechanisms have been developed to manage data-driven task offloading in MEC nodes.

Wang et al. [11] suggested a management mechanism for investing similarity between service requests. They have assumed similar tasks would require almost the same data from the same sensors. Therefore, they have suggested reorganizing the original tasks according to each related sensor in order to reduce the translation time and cost. However, in this study, the fog layer was viewed as a relay layer between the application layer and the cloud layer, despite this layer having some resources, and it can execute some tasks, particularly data-driven tasks, that require data to be passed through this layer before reaching the cloud. This work [12] suggests an offloading mechanism for computation-intensive applications either in Vehicle Edge Computing (VEC) or in Roadside Units (RSUs). While this work considers making offloading decisions for extensive computational applications (e.g., autonomous driving and vehicular video stream), data access in each node has not been considered. Nguyen et al. [13] focused on computation-intensive tasks that are generated by vehicles. Offloading decisions have been made based on priority, urgency, channel gain, and distance. However, the amount of data availability in each node has been overlooked in this study. Li et al. [1] developed a theoretical contract-based offloading paradigm from communication and computing perspectives. The paradigm focuses on compute-intensive and delay-sensitive tasks. The results have shown that the paradigm reduces system delay and energy consumption. However, tasks’ demands in each node were not taken into account during offloading decisions. Zhang et al. [14] proposed a theoretical contract approach to execution decisions. The goal of this study is to divide the road into various segments, each with its own set of RSUs to help with task processing.

Ning et al. [13] concentrated on computation-intensive tasks generated by vehicles. While offloading decisions are made based on the value of the utility function, which comprised four factors: priority, urgency, channel gain, and distance; each one has its own set of evaluation criteria. The results reveal that the suggested model outperforms the benchmark approaches in terms of execution time. However, the types of tasks that are dependent on a specific amount of data have been overlooked in this study.

Li et al. [1] developed a theoretical contract-based offloading paradigm. The proposed paradigm has investigated from the communication and computing perspectives, with a focus on compute-intensive and delay-sensitive tasks. The proposed model minimizes system latency and energy usage, according to the results. However, the amount of data required by each task has been ignored.

Zhang et al. [15] has built an efficient prediction model to make task processing decisions either locally on the vehicle or remotely based on tasks’ length. However, this model depends on the processing time, processing cost, and communication service providers. Despite this work having covered the type of tasks that require certain files to be executed, it has not discussed the amount of data in these files and how they could affect the offloading decision.

Sonmez et al. [16] demonstrated a two-stage FL interference model. This model considers both application requirements and EC resource utilities. The first FL stage focuses on system utilities (e.g., CPU utilization), while the second stage focuses on application requirements (e.g., task length and sensitivity). The primary purpose of this research is to improve the response time and extend the battery life of end-user devices. However, this paper discussed the upload/download data rates for each task, but it has not studied the percentage of data availability in EC. Overall, most of these studies have mentioned
compute-intensive tasks, but the nature of these tasks has rarely been considered, e.g., their popularity and tasks demands have been ignored, with the exception of [15]. As far as we know, this is one of the few studies that have considered the common factors between tasks and AN in the management mechanism design to fill the gaps left by the mentioned works.

2.2. FL Inference System

FL has been utilized in offloading decision making. It has been defined by Welstead in [17] as set of rules and regulations which defines boundaries and tells us what to do to be successful in solving problems within these boundaries. This type of logic tries to imitate human behavior in making decisions by avoiding the strict boundaries between categories in contrast to crisp logic. FL depends on studying the degree of membership and belonging. Therefore, FL is considered as an excellent option to manage real-world uncertainty due to the rapidly changing in different scenarios models [18]. Therefore, it has been employed for solving online and real-time problems. Take for illustration, the decision-making process for a heater according to the weather, defining a person relative to an age group, and the security level in shopping or trad online. In our context, it has been applied in many studies in order to make execution decisions in MEC Models. Nguyen et al. [19] adapted FL to reduce the number of failed tasks resulting from transmission collisions and support real-time applications. The proposed FL system aims to determine where each task should be executed either by its own resources, a local edge server, or the cloud. The proposed decision-making model is based on a two stages. The first stage focuses on where to place the incoming task, while the second stage determines the task processing place. However, this study has focused primarily on the resources availability while data assessment and task popularity are ignored.

Almutairi et al. [20] used the FL system to make the processing decisions for tasks either in the edge or in cloud. This work has implemented an FL based on three parameters: CPU utilization, WAN bandwidth, and delay sensitivity. This model provides much better results in terms of the average unsuccessful tasks and resources utilization compared to benchmark algorithms. However, even though the tasks sensitivities has been considered, data accessing is excluded in this work. Generally, FL has been discussed in many studies in order to improve MEC QoS. However, to the best of our knowledge, using tasks popularity and data overlapping as inputs to FL systems to improve MEC performance has not been studied yet for this domain.

3. System Model

This section provides an overview of the definitions of data-driven tasks, including their applications and challenges. Then, the proposed architecture is presented.

3.1. Data-Driven Tasks

The concept of data-driven tasks has drawn increased attention in the last few years. This term refers to tasks that rely heavily on raw data (e.g., text, photos, videos, medical data, weather information, etc.) that is generated by smart devices such as sensors and smartphones. These data are used by MEC/Cloud servers for knowledge building and decision-making purposes. As mentioned in [21], the core of data-driven tasks is data analysis. Noteworthy, data-driven analytic holds unlimited potential for assisting various domains of real-world scenarios such as dealing with air pollution, climate change, oil spill management, moving target tracking, healthcare monitoring, hazard analysis, real-time monitoring of stochastic damage in aerospace structures, forest fire propagation prediction, volcanic ash propagation, and traffic jams prediction could be precisely predicted according to data that have been collected by edge devises. However, there are challenges related to data-driven tasks. The data-driven tasks are considered to be compute-intensive and very complicated for edge devices. Hence, offloading a task for unsuitable MEC nodes would negatively affect QoS [22]. Additionally, in such types of tasks, the value of a subset of sensors and data may vary quickly due to the dynamic nature of the environment.
To overcome this issue, sampling rates methods should be considered. On the other side, sampling rates result in load imbalance and bottleneck problems. To address this challenge, we investigate data overlapping between tasks and nodes according to a query formulation, as we will elaborate later.

3.2. Service Architecture

Our system under consideration consists of a three-layer service architecture for data-driven tasks as shown in Figure 1: the sensing device layer, MEC layer, and Cloud computing layer. In the Sensing/Device, data are generated by unlimited devices (e.g., sensors and smart devices). For example, sensors that are spreading around specific area to collect data and transmit it to the MEC layer [11]. The MEC layer is a computing layer that sits between the sensing device layer and the cloud layers, which includes a set of collaborative ANs, such as vehicles, which are used to support dynamic environments providing, e.g., computing services for pedestrians (mobile users), traffic congestion services in smart cities, and computationally intensive, real-time, and delay sensitive applications [16]. When ANs receive the data generated by the sensing device layer, they store them locally until data accessing pattern (data overlapping) is required. Meanwhile, ANs also could execute some data-driven tasks locally according to tasks’ size, time constraint, popularity, data accessing, and resources availability and make a decision for other tasks, either to execute them locally and offload them to another AN or the cloud [12]. The Cloud layer has unlimited computation resources and, thus, tasks can be offloaded through, e.g., Base Stations (BSs) from the MEC layers due to limited resources or failure in completing tasks. However, determining the best layer for each task execution depends on the adopted task management mechanism and certain analytic application criteria.

![Figure 1. A three-layer architecture of EC ecosystem [11].](image-url)
4. Problem Fundamentals

In this section, we present the problem statement as follows: how do we efficiently assign data-driven analytic tasks to MEC/Cloud to minimize the task execution delay and increase MEC resource utilization. Then, we introduce the tasks’ management factors that have been adopted in this study.

4.1. Problem Statement

We consider the ANs-enable MEC system with a set of \( N \) ANs, denoted by \( N = \{ n_1, n_2, \ldots, n_N \} \). Each AN \( n_i \) collects \( N_i \) real-valued contextual data points denoted by \( x = [x_1, x_2, \ldots, x_d]^T \in \mathbb{R}^d \), with \( d \)-dimensional points, where each dimension refers to a specific feature (e.g., temperature, humidity). The \( n_i \) node stores them locally in dataset \( D_i = \{x_k\}_{k=1}^{N_i} \). Each node \( n_i \) has a neighborhood \( N'_i \subseteq N \) of directly communicating nodes \( n_j \in N'_i \). Moreover, node \( n_j \) communicates with the end-users/applications and the cloud. ANs can execute locally certain analytic tasks because they are equipped with specific computing resources. However, such resources might be limited for some tasks; thus, any decision of executing locally or offloading the tasks should be made carefully. Therefore, each node \( n_i \) needs to obtain certain information regarding the analytic tasks based on the following essential factors. First, the rate of tasks requests: A node \( n_i \) monitors the number of requests for each analytic task (\( T_k \)) coming with different request rate (\( \lambda_k \)) from end-users (applications). Based on this request rate \( \lambda_k \) per task, node \( n_i \) can assess which of the requested tasks is most popular. Evidently, popular tasks are preferred to be locally executed, hence avoiding further delays. This also helps to predict future tasks requests based on the current demands and their popularity. Moreover, node \( n_i \) can store the popular tasks in order to re-use them in the future thus potentially reducing the response time and resource consumption for future requests, while less popular tasks could be offloaded to node’s neighborhood \( n_j \) or to the cloud. Second, according to the task request rate \( \lambda_k \), some tasks are either extremely popular (in comparison to other tasks) or extremely rare. These are referred to as outlier tasks. Each node \( n_i \) can locally identify its own outlier tasks as it will be elaborated later. Generally, outlier tasks with high popularity (i.e., tasks are highly demanded) will be locally executed. On the other hand, outlier tasks with very low popularity can be offloaded to the available neighboring node \( n_j \) or, if none are available, they are offloaded to the cloud. The non-outlier tasks have all three options (local execution, offloading to available neighbor(s), or to the cloud). Third, as previously stated, each node \( n_i \) collects real-valued data and stores it locally in \( D_i \). It is worth noting that the type and amount of data in each node \( n_i \) significantly impacts whether a task should be locally executed or offloaded. Since we primarily focus on analytical tasks (e.g., ML model training and inference), such tasks require a specified amount of data from \( D_i \) to be executed. Specifically, consider a series of tasks \( T_1, T_2, \ldots, T_n \) arrive in node \( n_i \), which are treated in a queue until their execution or offloading decision is made. Such tasks have specific demands including the amount of data being accessed in order to be executed. Imagine for instance an analytic task as a series of value-range queries, which define a specific data subspace over the node \( n_i \)’s availability data in \( D_i \). In these cases, analytic task \( T_k \) might need a huge amount of the available data (e.g., >90% of the data) found in node \( n_i \), while only few data (e.g., >10%) are available in another node \( n_j \). Consequently, offloading such data-driven task to node \( n_j \) may demand extra time, resources, and the transmission of the required amount of data from \( n_i \) to \( n_j \). Therefore, our mechanism considers the amount of accessible data required for a given task to make the offloading decision.

Hence, given an incoming task \( T_k \) at node \( n_i \), the node locally estimates the probability of this tasks to be outlier (based on a recent history of demand rates) and the percentage of available data required. This information is used by node \( n_i \) to come up with the first two decisions/actions: \( a_0 = ‘\text{local tasks execution}’ \) or \( a_1 = ‘\text{task offloading}’, \) and if action \( a_1 \) is selected, then node \( n_i \) should swiftly make decisions in terms of in which neighboring node \( n_j \in N'_i \) task \( T_k \) should offload (action \( a_{12} \)) or whether to offload that \( T_k \) to the cloud (action \( a_{13} \)).
4.2. Task Management Factors

In this section, we elaborate on the basic factors for the proposed task management mechanism to be used for inferring the right execution decision for each task $T_k$ on each node $n_i$ based on the factors introduced in the previous section, including the popularity of tasks and the corresponding data access availability.

4.2.1. Task Popularity

We first elaborate on a methodology that determines the popularity of a task $T_k$ in a specific node $n_i$ within a sliding time window of size $W$. Specifically, assume a discrete time domain $t \in T = \{1, 2, \ldots\}$. At each time instance $t$, node $n_i$ observes a number of demands from each task coming from end-users or applications. The demand of task $T_k$ is linked with a request rate $\lambda_k$ as requested by end-users (applications) and monitored within the time window (horizon) $W$. Hence, given node $n_i$, a series of tasks $\{T_1, T_2, \ldots, T_k, \ldots\}$ arrive with rate $\lambda_k$. The demands for each task $T_k$ in $W$ recent time instances are recorded in the task requests vector $v_k = (v_{t-1+W}, v_{t-2+W}, \ldots, v_t)$, where $v_{t-l+W}$ element indicates the number of the incoming requests of task $T_k$ by end-users to node $n_i$ at time instance $l = 1, \ldots, W$. The requests vector $v_k$ over time window $W$ plays a significant role in storing the recent historical trends of each task $T_k$’s demands which will be used for estimating the popularity of $T_k$ task in a node $n_i$.

To derive such popularity for $T_k$, node $n_i$ groups the corresponding task demands within the time window adopting lightweight unsupervised clustering. The clustering algorithm divides the task demands of $v_k$ into groups (clusters). Each cluster contains a set of demands that have similarities between them according to the tasks’ arrivals within the time window. Specifically, we adopt the Subtractive Clustering [23] since the number of the clusters derived cannot be known beforehand neither is the same over different time windows. Subtractive clustering derives a set $C$ of $|C|$ clusters over the demands of task $T_k$ across the most recent $W$ time instances. Each cluster is represented by a task demands clusterhead $C_{\ell} \in C$, $\ell = 1, \ldots, |C|$. The clusterhead will help us in estimating the demands density for the arriving task $T_k$ during time window $W$ and the amount of requests per cluster as will be elaborated below. Given these (recently historical) statistics, we can define the popularity of the task $T_k$ based in its demanding behavior within the time window. If the task is associated with relatively many clusters within the specific time horizon $W$, then it is (statistically) considered more popular than other tasks associated with less clusters. Moreover, the more clusters are derived from clustering, the higher the variability and amount of task demands with different rates occur during the time window.

Therefore, we define the cluster density, which indicates the amount of demands of the $T_k$ within a specific time duration. Specifically, consider the $\ell$-th cluster $C_{\ell}$, which maintains the demand values $v_j \in C_{\ell}$ and is represented by the clusterhead demand $c_{\ell}$. We then define cluster variance $\sigma_{\ell}^2 = \frac{1}{|C_{\ell}|} \sum_{v_j \in C_{\ell}} (v_j - c_{\ell})^2$. Hence, in turn, we introduce the cluster density $d_{\ell}$ as the amount of the task demand values being within a squared distance of the cluster variance from the corresponding clusterhead (centroid).

$$d_{\ell} = |v_j \in C_{\ell} : (v_j - c_{\ell})^2 \leq \sigma_{\ell}^2|.$$  

Density $d_{\ell}$ depicts the number of task demands that are in a distance from the centroid less than the deviation. The deviation $\sigma_{\ell}$ is adopted to define our strategy concerning whether we want to be very ‘strict’ and requires many demand values to be very close to the centroid in order to conclude a high density. Within a cluster, there are historical demand values for task $T_k$ observed recent $W$ time instances. We pay significant attention on the clusters exhibiting a high density around the centroid. This density is strong evidence that multiple task demand values are realized around the centroid. For aggregating the demand information that clusters convey, we define a weighting scheme delivering a high weight for clusters with a high density. Specifically, based on the derived clusters, the popularity
index $p_k$ for tasks $T_k$ is the linear combination of the derived clusters weighted by their normalized densities:

$$\tilde{d}_\ell = \frac{d_\ell}{\sum_{j=1}^{|C|} d_j}$$

and, thus, the popularity demand index $p_k$ for task $T_k$ within recent $W$ time instances across all the derived clusters is defined as follows.

$$p_k = \sum_{\ell=1}^{|C|} \tilde{d}_\ell c_\ell.$$ (3)

### 4.2.2. Outlier Tasks

We introduce the concept of task outlier, which will support the decision-making process in our mechanism. The classification of a task as an outlier is used to determine the statistical extreme (non usual) demands of this task within recent $W$ time instances. Such mechanism undertakes the responsibility of annotating some of the tasks as outliers according to the popularity compared with the other tasks on a node.

The outlier tasks are divided into two classes: outliers that have relatively very high popularity than the usual trend and outliers that have relatively very low popularity. Classify them into these two classes based on adapting a lightweight process and using the Median Absolute Deviation (MAD) around the Median (MAD) across the popularity indices of the tasks in $T$ requested in node $n_i$ in the least $W$ time instances.

The median of the popularity indices $\tilde{p}$ is used as a separating point between the high and low popularity tasks. Based on the popularity median $\tilde{p}$ over the popularity values \(\{p_1, \ldots, p_M\}\), we can then calculate the MAD of the tasks $T$.

$$\text{MAD}(T) = \text{median}_{k=1,\ldots,M}(|p_k - \tilde{p}|)$$

Given this statistic, we define the outlier indicator $I_k$ for task $T_k$ based on its popularity $p_k$ as follows.

$$I_k = \frac{|p_k - \tilde{p}|}{\text{MAD}(T)}.$$ (5)

Task $T_k$ is an outlier, if $I_k$ is greater than the empirically derived threshold $\phi = 2.5$, i.e., the outlier indicator is described as follows.

$$o_k = \begin{cases} 1 \text{ (outlier)} & \text{if } I_k \geq \phi \\ 0 \text{ (non-outlier)} & \text{if } I_k < \phi \end{cases}$$

Based on the outlier tasks identification and the associated popularity indices of these tasks, node $n_i$ can obtain more certain decisions, either locally executing a very popular (outlier) task or offloading a very low popular (outlier) task. Nonetheless, the amount of data required for those outlier tasks (and of course of all the tasks) will further help the node to proceed with a right offloading decision as it will be elaborated later. As an informal guideline, the outliers filter selects those tasks with high popularity while having high data overlap with nodes can select action $a_0$. In contrast, very low popular tasks with low data overlapping will select action $a_{12}$.

### 4.2.3. Task’s Data Overlapping

Given an analytic task $T_k$ on a node $n_i$, we introduce the concept of the data overlapping, which indicates an estimation of the percentage of data (out of the entire dataset $D_i$) required for executing analytic task $T_k$. We concentrate on analytics tasks such as training machine learning models for applications such as federated learning, which has become increasingly popular and helpful in recent years. In this context, for instance, data points $x$ in a node represent real-values, e.g., sensed data that have been collected from IoT devises. These data are the basis to determine how much is suitable for a task $T_k$ to
be executed locally in node $n_i$. Meanwhile, the availability of data that each task requires varies from node to node. Therefore, if a task $T_k$ offloaded to $n_i$ has only 20% of data that it needs to be executed, this means we need to bring 80% of data in order to execute this task locally. This could lead to increases in resource consumption and the response time growing up. Given the representation of an analytic task $T_k$ via a (range) selection query $q_k = [q_{1}^\text{min}, q_{1}^\text{max}, \ldots, q_{d}^\text{min}, q_{d}^\text{max}]$ over a data sub-space defined be the dataset $D_i$, we define data overlapping as the ratio of the data points satisfying task query $q_k$ out of the data points stored in node’s dataset. That is, a data point $x \in D_i$ satisfies the range query $q_k$ if the following statement $S(q_k, x)$ holds true.

$$S(q_k, x) \equiv (q_1^\text{min} \leq x_1 < q_1^\text{max}) \land \ldots \land (q_d^\text{min} \leq x_d < q_d^\text{max})$$  \hspace{1cm} (7)

Hence, the degree of data overlapping $u_k$ of task $T_k$ represented via query $q_k$ is defined as follows.

$$u_k = \frac{|x \in D_i : S(q_k, x) \equiv \text{TRUE}|}{|D_i|}$$  \hspace{1cm} (8)

5. Task Management Reasoning
5.1. Fuzzy Linguistic Modeling

Given a node $n_i$ receiving demands for the analytics task $T_k$ over the node’s dataset $D_i$, we obtain the corresponding popularity $p_k$, outlier $o_k$ and data overlapping $u_k$. In this section, we are introducing a reasoning mechanism that takes into consideration the above-mentioned factors to proceed with task offloading decisions by balancing between the task demands, nodes’ capability, and nodes’ data availability. In order to deal with decision making reasoning, FL inference has been adapted to handle the inherent uncertainty and approximation of these factors in dynamic environments. Since it is the most prevalent strategy for dealing with rapid change in uncertain systems [24]. This is achieved by adapted fuzzy inference rules over linguistic variables that can model this type of uncertainty. There are many advantages for such inference performed locally on a node. First, it can easily cope with multi-criteria decision-making models by incorporating multiple factors in the same model. Second, it is capable of dealing with uncertainty in a dynamic context without complex mathematical models. Third, its lightweight computational complexity provides an explainable decision-making methodology [16]. This explainability is based on linguistic variables that reflect the uncertainty derived from the values of the factors $p_k$, $o_k$, and $u_k$.

In this context, popularity (fuzzy variable) is associated with three linguistic fuzzy values {High, Medium, Low} reflecting a high, medium, and low value of popularity for a specific task. Similarly, data overlapping (fuzzy variable) is associated with the linguistic values {High, Medium, Low} reflecting a high, medium, and low values of data overlapping derived from the task’s query data subpace over node’s data space. The outlier indicator $o_k$ (as a fuzzy variable) takes two linguistic values {Yes, No} reflecting whether $T_k$ is an outlier or not, as depicted in Figure 2. Given a linguistic value linked to a fuzzy variable, a membership function $\mu: \mathbb{R} \rightarrow [0, 1]$ is defined in order to indicate the possibility that a value of the variable belongs, at certain degree, to the linguistic value. Specifically, given a data overlapping value $u_k = x$, we associate this value with the linguistic value high via membership function $\mu^H_u(x) \in [0, 1]$. For instance, if the data overlapping $u_k = 0.7$ for task $T_k$, then this can possibly be considered as high data overlapping with possibility $\mu^H_u(0.7) = 0.88$. We similarly define these membership functions for the rest of the linguistic values for all factors.
There are different membership functions forms that can be adapted for fuzzy based reasoning, such as trapezoidal, piecewise linear, singleton, triangular, and Gaussian [25]. In our context, we consider the triangular form to represent membership functions, which is considered as the most common form according to [19]. To summarize, we obtain the next sets of membership functions of the fuzzy linguistic values for task popularity $p_k$, data overlapping degree $u_k$, and outlier indicator $o_k$, respectively.

$$F_{u_k}(x) = \{\mu_{u_k}^L(x), \mu_{u_k}^M(x), \mu_{u_k}^H(x)\}$$

$$F_{z_k}(x) = \{\mu_{z_k}^L(x), \mu_{z_k}^N(x), \mu_{z_k}^H(x)\}$$

$$F_{s_k}(x) = \{\mu_{s_k}^L(x), \mu_{s_k}^M(x), \mu_{s_k}^H(x)\}$$

5.2. Two-Stage Fuzzy Logic-Based Reasoning

Given the set of membership functions, we introduce a novel two-stage FL reasoning engine that makes the decision of task execution locally (actions $a_0$), offloading to another node $n_j \in N_i$ (action $a_{11}$) or offloading to the cloud (action $a_{12}$), as in Algorithm 1. Handling all these decisions in a single-stage FL is a complicated operation [16]. Therefore, we have adapted a two-stage FL system in order to reduce the system’s complexity. The first inference stage $S_1$ deals with the decisions (actions) $a_0$ = ‘local task execution’ and $a_1$ = ‘task offloading’. The output of $S_1$ is the offloading probability for a task $T_k$ given input $p_k$, $u_k$, and $o_k$, as will be elaborated later in this section. The second inference stage $S_2$ is based on $S_1$’s output. In particular, if $a_1$ action is selected (having the highest probability), then node $n_i$ swiftly decides in which neighboring node $n_j \in N_i$ task $T_k$ should be offloaded (action $a_{11}$), or it offloads $T_k$ to the cloud (action $a_{12}$).

The proposed two-stage reasoning mechanism runs on a specific node $n_i$, which plays the role of the ‘leader’ in neighborhood $N_i$. This role is periodically assigned to nodes from the neighborhood when certain criteria are met, e.g., remaining energy, computational capacity, and communication availability. This assignment is achieved via certain leader election mechanisms. We do not elaborate on these mechanisms, since it is beyond the scope of this paper. In the remainder, for the simplicity of notation, we assume that node $n_i$ is assigned with this leadership role to execute the two-stage reasoning engine, where all neighboring nodes $n_j \in N_i$ directly communicate with their leader $n_i$. Both FL system stages essentially follow the same steps, with the number of tasks varying. In particular, the first stage deals with all tasks, while the second stage only works with the set of tasks that could not be executed locally.

**Figure 2.** Implementation of FL on our three factors.
Algorithm 1 Two-stage FL inference system pseudocode

**Input:** $T_N$ with their parameters $(p_k, o_k, u_k)$

**Output:** Select the right Computational Resources (locally $n_i$, $n_j \in \mathcal{N}_i$, The cloud)

1. for All task in $T_N$ do
2. Read all $n_j \in \mathcal{N}_i$
3. Execute $S_1 \Rightarrow F = \text{FuzzyLogicSystem}(p_k, o_k, u_k)$
4. if $r_k \leq f_{\text{Low}}$ then
5. Allocate $t_k$ on $n_i$
6. else
7. for All task $t_k \in T_1$ do
8. Execute $S_2 \Rightarrow F_2 = \text{FuzzyLogicSystem}(p_k, o_k, u_k)$
9. if $r_k \leq f_{\text{Low}}$ then
10. Allocate $t_k$ in $n_j \in \mathcal{N}_i$
11. else
12. if $r_k \leq f_{\text{Medium}}$ AND $n_j \in \mathcal{N}_i$ (available) then
13. Allocate $t_k$ on $n_j \in \mathcal{N}_i$
14. else
15. Allocate $t_k$ to the cloud
16. end if
17. end if
18. end for
19. end if
20. end for

5.2.1. First Stage Reasoning S)

The $S_1$ reasoning engine on $n_i$ for each task $T_k$ goes through the following steps: The first step of FL is fuzzification of the inputs $(p_k, o_k, u_k)$ into their fuzzy linguistic terms via the membership functions as in Figure 3. It takes all these factors as numerical values (crisp values), then it assigns each value to the corresponding fuzzy values (e.g., Low, Medium, High) [19,20]. The second step is the activation of the Fuzzy Inference Rules (FIRs), which interpret the logic behind the decision making for the offloading probability. The obtained fuzzy values are then used to activate a set of FIRs, a.k.a., fuzzy knowledge base. Each FIR is represented via an IF-THEN statement [16]. The antecedent part (‘IF’ part) is a set of logical conjunctions over fuzzy linguistic variables. The consequent part (‘THEN’ part) of FIR is a fuzzy term from the set of linguistic terms {Low, Medium, High} that expresses offloading probability $r_k$. The generic format of FIR statements used in our $S_1$ engine is as follows:

$$\text{IF } p_k \text{ IS } X_1 \text{ AND } o_k \text{ IS } X_2 \text{ AND } u_k \text{ IS } X_3 \text{ THEN } r_k \text{ IS } X_4$$  \hspace{1cm} (10)

where the linguistic terms are $X_1, X_3, X_4 \in \{\text{Low, Medium, High}\}$ and $X_2 \in \{\text{No, Yes}\}$. For instance, the following FIR is described.

$$\text{IF } p_k \text{ IS HIGH AND } o_k \text{ IS YES AND } u_k \text{ IS HIGH THEN } r_k \text{ IS LOW.}$$

This rule expresses the decision of task $T_k$ to be offloaded with low probability, i.e., action $a_0$ is preferred more than action $a_1$, due to the fact that this task has very high popularity (thus being also an outlier) and the data required by this task can be fully available to node $n_l$ (high degree of overlapping). Hence, in this case, $T_k$ can be locally executed on node $n_l$ and not be offloaded (i.e., low offloading probability). Our $S_1$ engine requires 18 FIRs in the fuzzy knowledge base in order to cover the entire decision space; there are $3 \times 2 \times 3 = 18$ membership functions involved in three fuzzy variables: popularity, outlier,
and data overlapping. The FIRs of S1 engine are provided in Table 1, which reflects the reasoning behind the decision on actions \( a_0 \) or \( a_1 \) represented via the offloading probability.

![Figure 3. Fuzzy logic system working mechanism.](image)

**Table 1. FL rules inputs and the expected outputs.**

| \( R_i \) | \( p_k \) | \( a_k \) | \( u_k \) | \( r_k \) |
|---|---|---|---|---|
| 1 | Low | Yes | Low | High |
| 2 | Low | Yes | Medium | Medium |
| 3 | Low | No | Low | High |
| 4 | Low | No | Medium | High |
| 5 | Low | No | High | Medium |
| 6 | Medium | Yes | Low | Medium |
| 7 | Medium | Yes | Medium | Medium |
| 8 | Medium | Yes | High | Low |
| 9 | Medium | No | Low | High |
| 10 | Medium | No | Medium | Medium |
| 11 | Medium | No | High | Medium |
| 12 | High | Yes | Low | Medium |
| 13 | High | Yes | Medium | Low |
| 14 | High | Yes | High | Low |
| 15 | High | No | Low | Medium |
| 16 | High | No | Medium | Low |
| 17 | High | No | High | Low |
| 18 | High | No | High | Low |

The last step of S1 is the defuzzification of all the offloading probability values of the activated FIRs (16,19), which results in a scalar probability \( r_k = P(a_1) \) for the task \( T_k \). There are certain defuzzification operators for deriving scalar output over activated FIRs. We adapt the centroid defuzzifier, which not only is considered as the most common operator but also the defuzzified value that directly represents probability, which is aligned with the notion of \( r_k \) and calculated as follows:

\[
r_k = \frac{\int_{x \in [0,1]} x \mu_{r_k}^\nu(x)}{\int_{x \in [0,1]} \mu_{r_k}^\nu(x)},
\]

where \( \nu \) represents the \{Low, Medium, High\} linguistic terms of the offloading probability. The defuzzified offloading probability \( r_k \) ranges between 0% and 100%. In order to transform this probability to a decision, as Figure 4 depicts, we define the decision threshold to be 30%; that is, if \( r_k = P(a_1) \leq 0.3 \) (i.e., \( P(a_0) > 0.7 \)), then node \( n_i \) locally executes task \( T_k \). This means \( T_k \) will be processed locally by leader \( n_i \).
5.2.2. Second Stage Reasoning S2

The second stage of inference is introduced to deal with decision making on those tasks which have been determined to be offloaded as suggested by the S1 inference engine, i.e., their probability of offloading \( r_k > 0.3 \). After finishing S1, leader node \( n_i \) has two types of tasks: those that should be locally executed belonging to the set \( T_0 \subset T \) (associated with the action \( a_0 \)), and tasks belong to set \( T_1 \subset T \) (associated with the offloading action \( a_1 \)). The aim of the S2 inference engine is to proceed with decisions over the tasks in \( T_1 \) under the following actions: \( a_{10} \) (offload to another node) or \( a_{11} \) (offload to the cloud). Hence, S2 passes through two steps: tasks information updating and determine offloading probability.

If leader node \( n_i \) has not had enough resources for S2 inference, then another node \( n_j \in N_i \) can be elected as a leader according to specific leader election mechanisms. In this case, the old leader \( n_i \) just sends only the tasks \( T_1 \) to the new leader \( n_j \) in order to make offload decisions. Tasks information updating:

Leader node \( n_i \) collaborates with its neighbors to update the information regarding tasks in \( T_1 \) based on the S2 engine. For each task, a neighboring available and suitable node can be assigned based on the following reasoning. Firstly, leader \( n_i \) considers the task contextual information \((p_k, o_k, u_k)\) for each task \( T_k \in T_1 \) from each neighboring \( n_j \in N_i \). The goal is to determine how popular a task \( T_k \in T_1 \) is and how much data access it requires in node \( n_j \). Once \( n_i \) receives the request from leader \( n_i \), it sends over \((p_k, o_k, u_k)\) for each task in \( T_1 \) according to its dataset \( D_j \). In turn, \( n_i \) makes comparisons between tasks information and neighbor’s tasks information. When \( n_i \) receives information from \( n_j \), it then has two tables: the main table that it obtains from S1 and a new one that it receives from \( n_j \). It then updates the \( T_1 \) tasks information from \( n_j \) based on the following rule given a task \( T_k \in T_1 \).

**IF** \( p_{k,j} > p_{k,i} \) OR \( u_{k,j} > u_{k,i} \) **THEN**

\[
(p_{k,i}, o_{k,i}, u_{k,i}) \rightarrow (p_{k,j}, o_{k,j}, u_{k,j})
\]  

(12)

The rule states that task \( T_k \)’s information \((p_{k,i}, u_{k,i})\) in the leader node will be updated if the corresponding values from the neighbouring \( n_j \) are greater; otherwise, the task’s information will not be updated. The rationale behind updating tasks information is based on achieving lower \( r_k \). Therefore, if \( p_{k} \) and \( u_{k} \) are not greater than the ones in the leader node, this means that \( r_k \) will increase, and this leads to an increase in the probability of offloading the task to an unsuitable node or to the cloud. In order to avoid this, they will not be updated. This process will be repeated for all tasks obtaining information form each
neighboring node $n_j \in N_i$ sequentially. Hence, with each received task information from the next node $n_{j+1} \in N_i$, if the rule is fired, the leader’s task information keeps updating. By the end of this process, leader $n_i$ will have updated all the required information as shown in Table 2 according to the most suitable node.

### Table 2. $T_k$ Information updating according to suitable node.

| Task | Old Information | Update to | New Information |
|------|-----------------|-----------|-----------------|
| T1   | 8.37, Yes, 15%  | n2        | 9.64, No, 31.5% |
| T3   | 2.69, Yes, 45.5%| n4        | 2.01, Yes, 85.4% |
| T4   | 14.8, Yes, 67.8%| n2        | 13.71, No, 52.89% |
| T6   | 7.563, Yes, 31.8%| n3       | 5.88, No, 52.88% |
| T7   | 14.848, Yes, 15.67%| n2    | 26.45, No, 70% |
| T10  | 8.5, Yes, 21.3%  | n4        | 7.25, No, 66.2% |

Determine offloading probability: Even though the leader has updated with the most suitable node for each task, there is still a need to execute the $S_2$ fuzzy inference engine for those tasks in $T_1$, since this updating process only determines the best place for a task across neighboring nodes $n_j \in N_i$ regardless of $r_k$. Meanwhile, $S_2$ is applied in order to obtain a specific $r_k$ for each task in $T_1$. If $r_k$ is low, (action $a_{11}$) is decided, otherwise, (action $a_{12}$) is preferred.

Finally, the updated task information will be treated as input for $S_2$, and it will pass through the same steps as in $S_1$, i.e., fuzzification, the activation of FIR and defuzzification. By introducing $S_2$, it helps leader $n_i$ to decide clearly where each task $T_k$ should be executed according to the corresponding $r_k$ comparing with $S_1$ inference engine.

### 6. Experimental Evaluation

In this section, we used synthetic datasets to simulate tasks’ popularities, while the data overlapping experiment has been carried out on real datasets using analytics queries. Finally, the CloudSim Plus simulator has been used to measure the impact of our mechanism on upload/download data rate for each task and resource utilization.

#### 6.1. Tasks’ Popularities and Data Overlapping Experimental Setup

In this context, we deal with two types of datasets: real datasets and synthetic datasets. The real datasets are collected by four Unmanned Surface Vehicles (USVs) working as nodes $n_i$ to collect data from sensors in a coastal area (http://www.dcs.gla.ac.uk/essence/funding.html#GNFUV, accessed on 20 October 2020). Each USV node $n_i$ has a neighborhood $N_i \subset \mathcal{N}$ of directly communicating nodes $n_j \in N_i$. Moreover, node $n_i$ communicates end-users/applications in order to collect data and store them locally in their datasets $D_i$ for predictive analytic tasks. These data includes two features: sea surface temperature and humidity, i.e., $x = [x_1, x_2]^T \in \mathbb{R}^d$. There is one node $n_j \in \mathcal{N}$ acting as the leader that receives a set of analytic tasks $T_k$ and follows a specific mechanism in order to decide the following: whether $T_k$ should execute locally (action $a_0$) or offload (action $a_1$). We have assumed that leader $n_i$ has received ten analytic tasks and is investigating three factors ($p_k, o_k$, and $u_k$) for each task $T_k$. Regarding popularity ($p_k$), we have generated a synthetic dataset of task demands for each task $T_k$ according to different rates ($\lambda_k$) during a time window of size $W = 150$ by using Poisson distribution. Poisson distribution is considered as common tool to generate set of requests according to specific rate. After constructing the requests vector $v_k = (v_{t-1+W}, v_{t-2+W}, \ldots, v_t)$ for each task $T_k$, we have adapted the Subtractive clustering algorithm over tasks $T_k$ demands in order to group the demands according to the similarity between them. Then, the cluster density $d_t$ for each cluster $C$ has been calculated according to Equation (1). The leader $n_i$ obtains $T_k$’s popularity, as shown in the second column Table 3.
Regarding the outliers indicators $o_k$, we use the statistical threshold $\phi = 2.5$, where $n_i$ can generate the two sets of outliers and not outliers, as shown in the third column in Table 3. In order to obtain the task’s data overlapping $u_k$, we have defined for each local dataset $D_i$, the feature boundaries max and min values: $D_i = [x_1^{\min}, x_1^{\max}, x_2^{\min}, x_2^{\max}]$. Then, we generated queries $q_k$ uniformly at random ten tasks such that $q_k = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]$ for each task $T_k$ in order to obtain the data subspace needed for the execution of analytic task $T_k$, as shown in Table 4.

Evidently, there are some tasks $T_k$ with high data overlapping (e.g., $T_5$); $u_k$ reaches 98%, while there are tasks with low $u_k$, such as $T_3$ and $T_7$. Therefore, by executing tasks with high $u_k$ such as $T_5$ locally, it is expected to reduce the percentages of data offloading from 100% to 2%. In contrast, by executing tasks with low $u_k$ locally such as $T_7$, it is expected to increase data offloading percentages to almost 95%, which is obviously inefficient.

The FL engine has been developed in MATLAB considering the popularity $p_k$ of tasks $T_k$ between [1, 40] and outlier $o_k$ either 0 or 1, while the percentages of data overlapping $u_k$ are between [0%, 100%]. All these are inputs to the FL system, while the probability of offloading $r_k$ is the output in [0%, 100%]. As shown in Figure 5A, increasing $p_k$ and $u_k$ for task $T_k$ leads to a decrease in the probability of offloading $r_k$. This implies an increase in the probability of executing this locally (action $a_o$). On the other hand, in Figure 5B, decreasing $p_k$ and $u_k$ for task $T_k$ leads to increasing the probability of offloading $r_k$. This means that increasing the probability of offloading $T_k$ either to another node (action $a_{11}$) or to the cloud (action $a_{12}$).

### Table 3. Tasks $T_k$ information.

| $T_k$ | $P_k$ | $o_k$ | $u_k$ |
|-------|-------|-------|-------|
| $T_1$ | 8.37  | No    | 67.8% |
| $T_2$ | 28.31 | No    | 15.67% |
| $T_3$ | 2.69  | Yes   | 81%   |
| $T_4$ | 14.84 | No    | 22.88% |
| $T_5$ | 29.97 | No    | 6.8%  |
| $T_6$ | 7.56  | Yes   | 31.8% |
| $T_7$ | 26.84 | No    | 21.3% |
| $T_8$ | 39.49 | Yes   | 69%   |
| $T_9$ | 34.39 | Yes   | 45.5% |
| $T_{10}$ | 8.5 | No    | 45.5% |

### Table 4. Queries generation and percentages of data overlapping.

| Task | $q_1^{\min}$ | $q_1^{\max}$ | $q_2^{\min}$ | $q_2^{\max}$ | Points Including | Percentage |
|------|---------------|---------------|---------------|---------------|------------------|------------|
| $T_1$ | 19            | 32            | 49            | 57            | 130/899          | 14.46%     |
| $T_2$ | 19            | 29            | 44            | 46            | 164/899          | 18.24%     |
| $T_3$ | 26            | 28            | 43            | 58            | 75/899           | 8.3%       |
| $T_4$ | 22            | 32            | 42            | 53            | 160/899          | 17.79%     |
| $T_5$ | 20            | 32            | 38            | 58            | 885/899          | 98.44%     |
| $T_6$ | 20            | 29            | 41            | 55            | 310/899          | 34.48%     |
| $T_7$ | 21            | 25            | 48            | 53            | 48/899           | 5.33%      |
| $T_8$ | 22            | 33            | 38            | 55            | 600/899          | 66.74%     |
| $T_9$ | 20            | 32            | 50            | 57            | 470/899          | 52.28%     |
| $T_{10}$ | 19          | 28            | 36            | 50            | 251/899          | 27.91%     |
For task information update, we deal with ten tasks $T_k$, six of them should be offloaded (action $a_1$) according to Table 5. In order to determine the most suitable node $n_j \in N_i$ for each task $T_k$, information ($p_k$, $o_k$, and $u_k$) is updated.

During $S_2$, leader $n_i$ will have the task information as shown in Table 5. The output of $S_2$ is $r_k$ for each task in $T_1$. $S_2$ is applied with the same steps as in $S_1$. According to the results, $T_1$, $T_3$, $T_6$, and $T_7$ will offload to $n_j \in N_i$ (action $a_{11}$), while $T_4$ will offload to $n_j \in N_i$ (action $a_{11}$), if there are available resources. However, task $T_4$ and $T_{10}$ have almost very high offloading probability; therefore, they will be offloaded to the cloud (action $a_{12}$).

Table 5. $S_2$ decision making based on three factors.

| Task | Node | $P_k$ | $o_k$ | $u_k$ | $r_k$       | Rule          |
|------|------|-------|-------|-------|-------------|---------------|
| T1   | $n_2$| Low   | No    | Low   | 82%, High  | rule 4        |
| T3   | $n_4$| Low   | Yes   | High  | 49%, Medium| rule 3        |
| T4   | $n_3$| Medium| No    | medium| 58%, Medium| rule 11       |
| T6   | $n_2$| Low   | No    | Medium| 84.4%, High| rule 5        |
| T7   | $n_3$| High  | No    | Medium| 44%, Medium| rule 11       |
| T10  | $n_2$| Low   | No    | Medium| 51%, Medium| rule 5        |

6.2. Simulation Setup

CloudSim Plus has been utilized to create the considered scenarios and to evaluate the performance of our mechanism. In this experiment, two types of parameters have been considered: data-driven task characteristics and MEC/cloud parameters. Data-driven tasks characteristics vary according to the nature of tasks. Some tasks are affected by delays, while others are not; some tasks could execute on MEC nodes, while others are beyond MEC node’s capabilities and should be offloaded to the cloud. To simulate real-life scenarios, ten different data-driven tasks (applications) have been used. To decide the application types, we looked at the most common data-driven tasks (weather prediction, air pollution prediction, traffic jam prediction, compute-intensive tasks, and health apps, etc.). Table 6 contains tasks information chosen based on [16]. The upload/download data size...
represents the type of data sent/received from EC/Cloud since it could increase or decrease according to data overlapping percent, and this is what distinguishes our mechanism against other task offloading mechanisms. For instance, (50,000 MB, 100 MB) denotes the size of uploaded data (humidity, temperature, wind, etc.) that will be used to build a ML model, and downloads depict the model that the application will receive as a result of data collection and training in EC/cloud computing. According to our mechanism, if the data overlapping percentage is high (e.g., 90% or more) the uploading data could be reduced from 50,000 MB to 10 MB. Task length determines the number of Million Instructions (MI) and the required CPU resource to complete a data-driven task. We considered ten tasks arriving at \( n_i \) with specific features, which include task length and upload/download data. According to data overlapping, we made the range of this parameter fluctuate from low values with some tasks to high values with others, while resource consumption and task delay sensitivity have been set up according to the applications indicated in [16].

| Task | Application               | Task Length | Upload/Download Data |
|------|---------------------------|-------------|----------------------|
| T1   | Deep learning             | 10,000      | 50,000/100           |
| T2   | Traffic jam prediction    | 20,000      | 200,000/300          |
| T3   | Air pollution prediction  | 15,000      | 200,000/400          |
| T4   | Healthcare diagnosis      | 30,000      | 80,000/100           |
| T5   | Weather prediction        | 8500        | 50,000/50            |
| T6   | Compute-intensive task    | 20,000      | 300,000/500          |
| T7   | Fraud detection           | 18,000      | 300,000/250          |
| T8   | Virtual assistants        | 25,000      | 20,000/50            |
| T9   | Alerting And Monitoring   | 14,000      | 100,000/300          |
| T10  | Social Media Analysis     | 21,000      | 60,000/80            |

Other simulation parameters that reflect the computational capabilities of MEC/Cloud servers, such as bandwidth, the number of Virtual Machines (VM), and host MIPS, are listed in Table 7.

| Parameters          | EC                  | Cloud            |
|---------------------|---------------------|------------------|
| Bandwidth           | WAN 500 MB/s        | LAN 10 GB/s      |
| Number of VM        | 2                   | 8                |
| Number of cores     | 2                   | 8                |
| VM CPU speed        | 10 MB               | 100 MB           |
| HOST MIPS           | 1000                | 10,000           |

6.3. Comparative Assessment

In this section, we present two types of results that reflect our mechanism’s performance:

First, in terms of considering both factors (tasks polarities \( p_k \) and data overlapping \( u_k \)), we compare the suggested mechanism’s efficiency to the effectiveness of two alternative mechanisms that deal with the same tasks and datasets. The first mechanism (\( M_2 \)) takes only tasks popularity \( p_k \) and outlier \( o_k \) into consideration when it makes the offloading decision. The second mechanism (\( M_3 \)) only takes the percentages of data overlapping \( u_k \) between the tasks \( T_k \) and nodes \( n_j \in N_i \). The experimental results in Table 8 show the performance of the mechanism, coined here as (\( M_1 \)), which is the highest according to the optimal solution (OS) in the last column at the same table. As we can see, (\( M_2 \)) focuses on investigating task popularity \( p_k \) in each node, while data overlapping \( u_k \) is completely ignored. That means \( M_2 \) mechanism will distribute tasks \( T_k \) among \( n_j \in N_i \) regardless of whether task \( T_k \) is offloaded to the suitable node \( n_j \in N_i \) that could reduce the response time
and resource consumption or not. On the other hand, \((M_3)\) only focuses on the percentages of data overlapping \(u_k\). \((M_3)\) uses the node’s data in an efficient manner regardless of whether tasks \(T_k\) are popular or not. As a consequence, the popular/urgent tasks \(T_k\) could offload to the cloud (action 12) or could wait in the execution queue, because they have lower data overlapping \(u_k\) with \(n_j \in N_j\). \((M_1)\) attempts to balance between the both sides of popularity \(p_k\) and data overlapping \(u_k\). Based on the results, \((M_1)\) can provide an accurate offloading probabilities \(r_k\) close to 95% according to OS boundaries, while \((M_2)\) and \((M_3)\) provide accuracies reaching 90% and 60%, respectively.

**Table 8.** The probability of offloading \(u_k\) for each task \(T_k\) according to our mechanism compared to the other two mechanism.

| \(P_k\) | \(o_k\) | \(u_k\) | \(r_k\) | \(M_1\) | \(M_2\) | \(M_3\) | OS |
|---|---|---|---|---|---|---|---|
| T1 | Low | Yes | Low | 83% | 84% | 57% | High |
| T2 | Med | No | High | 30.4% | 32% | 42% | Med |
| T3 | Low | No | Low | 85% | 86% | 72.92% | High |
| T4 | Med | No | Low | 65% | 68.9% | 53% | Med |
| T5 | High | No | Med | 23% | 17.6% | 35% | Low |
| T6 | Low | Yes | Low | 85% | 86.5% | 72.7% | High |
| T7 | Med | No | Med | 44.7% | 37% | 47% | Med |
| T8 | High | Yes | High | 14.4% | 14.5% | 17.7% | Low |
| T9 | High | Yes | High | 15% | 15.8% | 27.1% | Low |
| T10 | Low | Yes | Low | 83% | 85% | 67.2% | High |
| - | - | - | - | 10/10 | 8/10 | 6/10 | - |

Second, in terms of resource utilization: We compare the effectiveness of our mechanism against two alternative mechanisms over the same task’s simulation conditions. The first one, cloud-based mechanism [11], where the MEC nodes have been used to collect sensors data and sent them to the cloud to reduce sensors’ energy consumption that would happen if data have been sent directly to the cloud. The second mechanism, a MEC-based mechanism, has been suggested in many studies, such as [16,20], where the tasks are sent to the MEC node that has the highest availability, bandwidth, and task delay sensitivity. Simulating our mechanism resulted in a high data uploading speed between one to ten minutes, while the uploading speed in the cloud-based model is between 28 to 60 min. whereas the uploading speed in MEC-based mechanisms, which has not considered data overlapping, is almost double the speed we obtained with our mechanism (see Figure 6a).

In terms of execution time, we have considered data offloading time in addition to the main execution time because, in the data-driven tasks, data are considered an integral part of the task execution. Figure 6b shows that the execution time is extremely minimized compared to the cloud-based model. Moreover, we can observe that the bandwidth is reduced as well. The results of cloud (WAN) and MEC (MAN) bandwidth measurements are shown in Figure 7. The blue line represents the bandwidth usage percent according to our mechanism, which is considered to be very low compared to the other mechanisms. The red line represents the bandwidth usage according to the MEC-based mechanism, which is considered almost double our mechanism usage. Meanwhile, the black line shows bandwidth usage in order to execute these ten tasks on the cloud, which is very high usage compared to ours and MEC-based (see Figure 7a).
According to resource usage, Cloud-based mechanism produced the best performance compared to our mechanism and EC-based mechanism as it has unlimited resources (Figure 7b).

7. Conclusions

In this work, we introduced a mechanism for data-driven analytics tasks in an AN-enabled MEC environment to exploit their resources efficiently and reduced the response time. In particular, the core of this mechanism focuses on three factors: task popularity, outlier, and data overlapping to make execution decisions for each task. Task popularity concentrates mainly on investigating each task’s demands, while the outlier determines the statistically extreme (non-usual) demands of tasks. Meanwhile, data overlapping studies the percentages of data overlapping between tasks and ANs. These three factors are treated
as input to a two-stage FL interference system to make the final decision for each task. Our mechanism performance has been evaluated according to the probability of offloading data-driven analytics tasks to the right nodes according to the optimal solution and against two other mechanisms. As evidenced by the results, our mechanism significantly outperforms the benchmark mechanisms in terms of decision-making accuracy. Furthermore, this mechanism can reduce the probability of a task being offloaded to an unsuitable node by up to 90%. Then, our method has been evaluated in terms of resource utilization, showing that it provides higher data uploading speeds compared to EC-based and cloud-based methods. In addition, bandwidth usage has been reduced dramatically compared to benchmark mechanisms. As a consequence, data-driven analytic task execution times have been minimized. Our future agenda include methods that are expected to improve our mechanism by considering the tasks’ delay sensitivity, energy consumption, and nodes’ mobility.

**Author Contributions:** Conceptualization, T.A., K.K., C.A.; investigation, T.A., I.A., C.A.; writing—original draft preparation, T.A., C.A., I.A.; writing—review and editing, T.A., C.A. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data can be accessed at, [http://www.dcs.gla.ac.uk/essence/funding.html#GNFUV](http://www.dcs.gla.ac.uk/essence/funding.html#GNFUV).

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

**Abbreviations**

The following abbreviations are used in this manuscript:

- $T_n$: Set of analytic tasks;
- $t$: Time instance;
- $V_k$: The task requests vector;
- $d_i$: Historical demand observations vector, $d = d_1, d_2, ..., d_W$;
- $W$: Time window of size;
- $C$: Cluster;
- $C_\ell$: Clusterhead;
- $\lambda_k$: Task demands rate;
- $v_j$: The demand values;
- $\sigma_j$: The deviation;
- $d_f$: The cluster density;
- $\bar{d}_f$: Densities normalization;
- $p_k$: Popularity;
- $\bar{p}$: The median of the popularity;
- $\text{MAD}$: Mean absolute deviation;
- $l_k$: The outlier indicator;
- $\phi$: An empirically derived threshold;
- $o_k$: Outliers;
- $p_k$: The degrees of popularity;
- $v$: Linguistic terms of the offloading probability;
- $T_0$: Set of task will execute locally;
- $T_1$: Set of task will offload;
- $u_k$: Data overlapping;
- $D$: Dataset in each AN;
- $x$: Data point;
- FL: Fuzzy logic;
- $q_k^{\text{min}}, q_k^{\text{max}}$: Maximum and minimum value for the temperature in a D;
\[
\begin{align*}
\theta^\text{min}_k, \theta^\text{max}_k & \quad \text{Maximum and minimum value for the humidity in a D;} \\
\eta_k & \quad \text{Data overlapping;} \\
\sigma_k & \quad \text{Outlier.}
\end{align*}
\]

**References**

1. Li, S.; Hu, X.; Du, Y. Deep Reinforcement Learning for Computation Offloading and Resource Allocation in Unmanned-Aerial-Vehicle Assisted Edge Computing. *Sensors 2021*, 21, 6499.
2. Budhiraja, I.; Kumar, N.; Tyagi, S.; Tanwar, S. Energy Consumption Minimization Scheme for NOMA-Based Mobile Edge Computation Networks Underlaying UAV. *IEEE Syst. J. 2021*, 15, 5724–5733.
3. Cheng, J.; Guan, D. Research on task-offloading decision mechanism in mobile edge computing-based Internet of Vehicle. *EURASIP J. Wirel. Commun. Netw. 2021*, 2021, 101.
4. Jodelka, O.; Anagnostopoulos, C.; Kolomvatsos, K. Adaptive Novelty Detection over Contextual Data Streams at the Edge using One-class Classification. In Proceedings of the 2012 12th International Conference on Information and Communication Systems (ICICS), Valencia, Spain, 24–26 May 2021; pp. 213–219.
5. Cui, K.; Lin, B.; Sun, W.; Sun, W. Learning-based task offloading for marine fog-cloud computing networks of USV cluster. *Electronics 2019*, 8, 1287.
6. Anagnostopoulos, C. Edge-centric inferential modeling & analytics. *J. Netw. Comput. Appl. 2020*, 164, 102696.
7. Kolomvatsos, K.; Anagnostopoulos, C.; Koziri, M.; Loukopoulos, T. Proactive amp; Time-Optimized Data Synopsis Management at the Edge. *IEEE Trans. Knowl. Data Eng. 2020*.
8. Kolomvatsos, K.; Anagnostopoulos, C. A deep learning model for demand-driven, proactive tasks management in pervasive computing. *IoT 2020*, 1, 240–258.
9. Kong, F.; Li, J.; Jiang, B.; Zhang, T.; Song, H. Big data-driven machine learning-enabled traffic flow prediction. *Trans. Emerg. Telecommun. Technol. 2019*, 30, e4382.
10. Kolomvatsos, K.; Anagnostopoulos, C. Proactive, uncertainty-driven queries management at the edge. *Future Gener. Comput. Syst. 2021*, 118, 75–93.
11. Wang, P.; Yu, R.; Gao, N.; Lin, C.; Liu, Y. Task-driven data offloading for fog-enabled urban IoT services. *IEEE Internet Things J. 2020*, 7, 7562–7574.
12. Nguyen, V.; Khanh, T.T.; Tran, N.H.; Huh, E.N.; Hong, C.S. Joint offloading and IEEE 802.11 p-based contention control in vehicular edge computing. *IEEE Wirel. Commun. Lett. 2020*, 9, 1014–1018.
13. Ning, Z.; Dong, P.; Wang, X.; Rodrigues, J.J.; Xia, F. Deep reinforcement learning for vehicular edge computing: An intelligent offloading system. *ACM Trans. Intell. Syst. Technol. (TIST) 2019*, 10, 1–24.
14. Zhang, K.; Mao, Y.; Leng, S.; Vinel, A.; Zhang, Y. Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In Proceedings of the 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM), Halmstad, Sweden, 13–15 September 2016; pp. 288–294.
15. Zhang, K.; Mao, Y.; Leng, S.; He, Y.; Zhang, Y. Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Veh. Mag. 2017*, 12, 36–44.
16. Sonmez, C.; Ozgovde, A.; Ersoy, C. Fuzzy workload orchestration for edge computing. *IEEE Trans. Netw. Serv. Manag. 2019*, 16, 769–782.
17. Welstead, S.T. *Neural Network and Fuzzy Logic Applications in C/C++*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1994.
18. Khoshkholgh, M.G.; Navaie, K.; Yanikomeroglu, H.; Leung, V.C.; Shin, K.G. Randomized caching in cooperative UAV-enabled fog-RAN. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019; pp. 1–6.
19. Nguyen, V.; Khanh, T.T.; Nguyen, T.D.; Hong, C.S.; Huh, E.N. Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications. *J. Cloud Comput. 2020*, 9, 66.
20. Almutairi, J.; Aldossary, M. A novel approach for IoT tasks offloading in edge-cloud environments. *J. Cloud Comput. 2021*, 10, 28.
21. Samea, F.; Azam, F.; Rashid, M.; Anwar, M.W.; Haider Butt, W.; Muzaffar, A.W. A model-driven framework for data-driven applications in serverless cloud computing. *PLoS ONE 2020*, 15, e0237317.
22. Chen, Q.; Zheng, Z.; Hu, C.; Wang, D.; Liu, F. On-edge multi-task transfer learning: Model and practice with data-driven task allocation. *IEEE Trans. Parallel Distrib. Syst. 2019*, 31, 1357–1371.
23. Rao, U.M.; Soed, Y.; Jarial, R. Subtractive clustering fuzzy expert system for engineering applications. *Procedia Comput. Sci. 2015*, 48, 77–83.
24. Sharma, N.; Magarini, M.; Jayakody, D.N.K.; Sharma, V.; Li, J. On-demand ultra-dense cloud drone networks: Opportunities, challenges and benefits. *IEEE Commun. Mag. 2018*, 56, 85–91.
25. Chen, Z.; Xiao, N.; Han, D. Multilevel task offloading and resource optimization of edge computing networks considering UAV relay and green energy. *Appl. Sci. 2020*, 10, 2592.