Data-Driven Analytics Task Management at the Edge: A Fuzzy Reasoning Approach

Tahani Aladwani*, Ibrahim Alghamdi†, Kostas Kolomvatsos‡, Christos Anagnostopoulos§
* § School of Computing Science, University of Glasgow, UK
† Faculty of Science and Arts in Baljurashi, Dept. of Computer Science, Al-Baha University, SA
‡ Dept. of Informatics and Telecommunications, University of Thessaly, GR
*2587711A@student.gla.ac.uk, †ia.alghamdi@bu.edu.sa, ‡kostasks@uth.gr, §christos.anagnostopoulos@glasgow.ac.uk

***Abstract*—Dynamic data-driven applications such as tracking and surveillance have emerged in Internet of Things (IoT) environments. Such applications rely heavily on data generated by connected devices (e.g., sensors). Consequently, leveraging these data in building data-driven predictive analytics tasks improves the Quality of Service (QoS) and, as a result, Quality of Experience (QoE). Such data support various data-driven tasks such as regression and classification. Analytics tasks require data and resources to be executed at the edge since transferring them to the cloud negatively affects response times and QoS. However, the network edge is characterized by limited resources compared to the cloud, being the subject of constraints that are violated upon offloading data-driven tasks to improper edge nodes. We contribute with an analytics task management mechanism based on the context of the requested data, the task delay sensitivity and the VM utilization. We introduce a novel Fuzzy inference mechanism for determining whether data-driven tasks should be executed locally, offloaded to peer edge servers, or sent to cloud. We showcase how our fuzzy reasoning mechanism efficiently derives such decisions by calculating the offloading probability per task. The derived optimal actions are compared against benchmark models in Edge Computing (EC) environments.**

***Index Terms*—Edge computing, data-driven tasks offloading, data-overlapping, Fuzzy inference system.**

I. Introduction

Autonomous driving, smart cities services, and Augmented Reality are just a few examples of new computational-intensive and data-driven applications over the IoT infrastructure [9]. 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 [16], [3] as the main solution to reduce the burden of data-driven tasks on edge devices. However, cloud computing is not the best option for delay-sensitive and dynamic data provisioning applications that should be completed under real-time constraints. This is due to the fact that data centres are typically located in distant places from data sources. As a result, data processing in the cloud will eventually require increased communication activities via wide-area networks (WANs). This increases the traffic in the network, the probability of tasks failures, and evidently, results in relatively high response times [5]. EC has already been playing a significant role in reducing these obstacles by bringing computing services close to end-users and data sources [6]. The adoption of the EC brings many benefits for applications, such as minimized delay, traffic reduction and increased bandwidth availability due to the minimization of the amount of data that should be transferred in an upwards mode to the cloud. Using EC to execute data-driven task has been overlooked in the most of data-driven tasks studies. In particular, EC has been adopted to support real-time execution and data analytics applications. As a consequence, end-users demand execution and delivery of data-driven tasks from EC [7],[9]. EC has limited capacity making the holistic task execution of all requests a challenge, especially, when these tasks involve 'intense' data processing (e.g., clustering and classification) [6]. As a result, implementing a data-driven task management mechanism is critical for distributing tasks among EC nodes and the cloud according to specific criteria like time constraints for getting a response and task data accessibility. The efficient management of all these criteria could dramatically increase the utilization of resources [8], reduce the response delay, and improve Quality of Service (QoS). Motivated by this challenge, this research focuses on the *data-overlapping* of each task with the available nodes (the percentage of data accessing). Because most, if not all, analytic tasks necessitate direct access to distributed datasets stored on EC nodes. Therefore, task offloading decisions should take into consideration the required data that need to be accessed by tasks. The data needed per task drives the decision-making on whether to execute a task locally, offload a task to neighboring EC or to the cloud based on their data availability. There are also other criteria that should be considered in the offloading decision-making process. The status of computation resources, e.g., Virtual Machine (VM) utilization, has a substantial impact on the QoS. To make our mechanism able to swiftly balance between these criteria and effectively handle the inherent uncertainty in the decision-making, we propose the adoption of a Fuzzy Inference process to implement decision-making rules based on the principles of the Fuzzy Logic (FL) [11]. FL is considered to be one of the most popular methods for dealing with the rapid change uncertain systems. Additionally, it has a lower computational complexity compared to other decision-making methods [15]. In our context, FL has been employed to determine the *probability of offloading* for each data-driven task according to the aforementioned criteria. Our main contributions are:

- An analytics task management decision-making mechanism based on data overlapping, task delay sensitivity, and VM utilization;
- A novel FL reasoning system that derives the probability of offloading per task;
- Comparative assessment and performance evaluation of our mechanism against alternative and baseline methods.

The paper is organized as follows: Section II elaborates on related work while Section III represents our system model. Section IV formulates our problem and Section V introduces our tasks management mechanism. Section VII reports on experimental evaluation while SectionVIII concludes the paper.

II. Related work

Due to resource, energy, and storage limitations in EC nodes, selecting a proper task management mechanism is crucial. Several mechanisms have been proposed to manage data-driven task offloading in EC. Wan et al. [16] have adopted Fog Computing architecture to develop a task-driven data offloading algorithm for urban IoT services. The suggested architecture relies on mobile gateways in the fog layer to collect data that are needed to complete a task. Then, only the required data will be uploaded to the cloud rather than uploading the whole data. This method helps in reducing sensors' energy consumption, decreasing data transmission cost, and resource and network consumption. However, in this study, the fog layer was viewed as a relay layer between the application layer and the cloud layer, despite the fact that this layer can perform some tasks, particularly data-driven tasks, which require data to be passed through this layer before reaching the cloud. Mukherjee et al. [10] have investigated horizontal collaboration between many nodes and vertical collaboration with the cloud for parallel task data offloading in order to reduces the overall latency for data-driven tasks. However, this mechanism has considered offloading data as one piece either to another node or to cloud, while the required data availability in each node has been ignored. Nguyen et al. [12] have suggested an offloading mechanism for computation-intensive applications either in Vehicle Edge Computing (VEC) or in Roadside Units (RSUs). While this work considers making offloading decision for extensive computational applications (e.g., autonomous driving and vehicular video stream), data access in each node has not been considered. Ning et al. [13] have focused on computation-intensive tasks generated by vehicles. Offloading decisions are based on priority, urgency, channel gain, and distance. Then, to arrive at an intelligent decision, deep reinforcement learning (DRL) is applied. The amount of data availability in each node has been overlooked in this study. Moreover, Li et al. [9] 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, the amount of data required by each task was not taken into account during offloading decisions. Our work focuses on local data overlapping, resource utilization and task sensitivity in order to fills the gaps left by the above efforts.

III. System Model

A. Data-driven Tasks

The concept of data-driven tasks has drawn increased attention during the last few years. Data-driven tasks refers to tasks that rely heavily on data generated by smart devices (e.g., sensors, smartphones) to build knowledge and make decisions. As mentioned in [14], the core of data-driven applications is data analysis. Various real-world applications in different domains become data-driven to make precise decisions. Consequently, domains 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 has 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 to unsuitable EC node would negatively affect the QoS [3]. 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 EC nodes according to a query formulation, as we will elaborate later.

B. Service Architecture

Our system under consideration consists of a three-layers service architecture for data-driven tasks as shown in Fig. 1: the application layer, EC layer and cloud computing layer. **In the Application layer**, data are generated by unlimited devices (e.g., sensors, smart devices). For example, sensors that are spreading around specific area to collect data (e.g., humidity and temperature) and transmit it to the EC layer via getaways adopting low-power wireless communication technology [16].

The EC layer is a fixed computing environment that sits between the application layer and the cloud layer, with some resources to execute relatively some data-driven tasks are generated by applications, since it considers close to data and tasks sources. This layer includes a set of collaborative EC nodes, which are used to provide real-time computing services e.g., traffic congestion services in smart cities, and computationally intensive, real-time, and delay sensitive applications. EC nodes receive the data generated by application layer and store them locally until data accessing pattern are required [16]. The decision of either to execute these tasks locally, offload it to another node or in the cloud can be made based on different factors such as task size, time, popularity, upload/download data, and resources availability in nodes [12], [15]. **The cloud layer** has unlimited computation resources and thus tasks can be offloaded through e.g., Base Stations (BSs) from the EC

layers due to limited resources or failure in completing tasks. Determining the best layer for each task execution depends on the adopted task management mechanism and certain analytics application criteria.

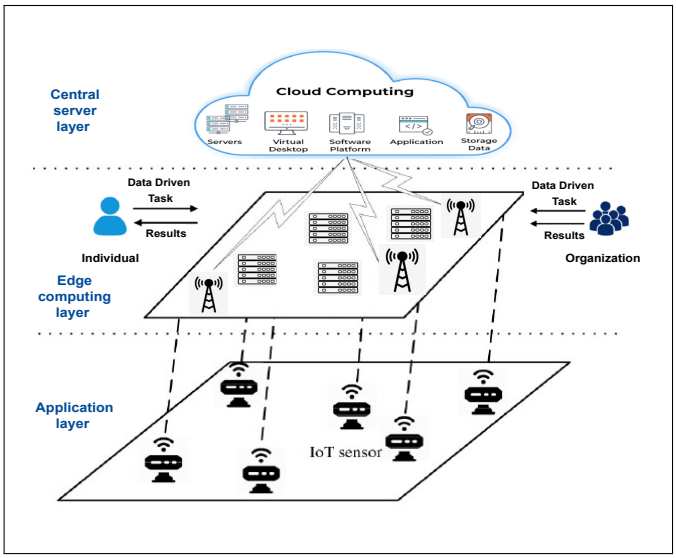

Fig. 1: A three-layers architecture of EC ecosystem.

IV. Rationale

We consider the EC system with a set of N EC nodes, denoted by $\mathcal{N} = \{n_1, n_2, \ldots, n_N\}$. Each EC n_i collects N_i real-valued contextual data points $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top \in \mathbb{R}^d$, with d-dimensional points, where each dimension refers to a specific feature (e.g., temperature, humidity). The n_i node stores them locally in the dataset $\mathcal{D}_i = \{\mathbf{x}_k\}_{k=1}^{N_i}$. Each node n_i has a neighborhood $\mathcal{N}_i \subset \mathcal{N}$ of directly communicating nodes $n_j \in \mathcal{N}_i$. Moreover, node n_i communicates with the end-users/applications and the cloud. EC nodes can execute locally certain data-driven tasks because they are equipped with specific computing resources. However, such resources might be limited for some tasks, and thus any decision of executing locally or offloading the tasks should be made carefully. Each node n_i needs to obtain certain information regarding the data-driven tasks based on the following essential factors. First, data overlapping, as previously stated, each node n_i collects real-valued data and stores it locally in $\mathcal{D}_i$. It's 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 $\mathcal{D}_i$ to be executed. Furthermore, we consider a series of tasks $T_1, T_2, ..., T_n$ 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 available data in $\mathcal{D}_i$. In this cases, the 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 a small amount of data (e.g., $>$10%) is available in another node n_j. Consequently, offloading such data-driven task to node n_j could evidently consume more time and resources transferring the required amount of data from n_i to n_j for such an execution. Therefore, our mechanism considers the amount of accessible data required for a given task to make the offloading decision as the main parameter. Also, two other parameters have been considered in this decision-making mechanism, which are *resources availability* and *delay sensitivity*. These two parameters are considered to be an essential aspect of any task management mechanism, regardless of whether the tasks are data-driven or not. All of these parameters will be explored in more detail in the following section. Hence, given an incoming task T_k at node n_i, node locally estimates the percentage of available data required, required resources, and task sensitivity. Node n_i uses this information to determine the execution mode, which is one of three decisions/actions. They are a_0 ='local tasks execution', a_1 ='task offloading', or a_{12}= offload to the cloud.

V. Task Management Mechanism Factors

We introduce the basic factors for inferring the right execution decision for each task T_k on each node n_k corresponding to data availability, given resources availability and task's delay sensitivity.

A. Task Data Overlapping

Given a data-driven 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 whole dataset $\mathcal{D}_i$) required for executing the analytic task T_k. We specifically focus on analytics tasks, such as training ML models for e.g., a federated learning mechanism, which is rather popular and useful in the recent years. In this context, for instance, data points x in a node represent real-values/sensed data that have been collected from IoT devises. These data are the basis for determining how much is appropriate 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 is offloaded to n_i and has only 20% of data it requires for execution, it means we need to bring 80% of data in order to execute this task locally. As a result, resource utilization and response time may increase. Given the representation of an analytic task T_k via a (range) selection query $\mathbf{q}_k = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]$ over a data sub-space defined be the dataset $\mathcal{D}_i$, we define data overlapping as the ratio of the data points satisfying the task query $\mathbf{q}_k$ out of the data points stored in node's dataset. That is, a data point $\mathbf{x} \in \mathcal{D}_i$ satisfies the range query $\mathbf{q}_k$ if the following statement $\mathcal{S}(\mathbf{q}_k, \mathbf{x})$ holds true:

$$\mathcal{S}(\mathbf{q}_k, \mathbf{x}) \equiv (q_1^{\min} \leq x_1 < q_1^{\max}) \wedge \ldots \wedge (q_d^{\min} \leq x_1 < q_d^{\max}) \quad (1)$$

Hence, the degree of data overlapping u_k of task T_k represented via the query $\mathbf{q}_k$ is defined as:

$$u_k = \frac{|\mathbf{x} \in \mathcal{D}_i : \mathcal{S}(\mathbf{q}_k, \mathbf{x}) \equiv \texttt{TRUE}|}{|\mathcal{D}_i|} \tag{2}$$

B. EC Resources Utilization

EC resources utilization represents the current level of utilisation of the VM hosted by the local edge server [1]. In this context, n_i is expected to perform multiple tasks at the same time, such as data collection, task execution, and acting as a release node between the application layer and cloud server layer, all of which affect EC resource availability [15]. Therefore, EC resources utilization is an essential aspect in task offloading decision. Accordingly, three decisions can be made according to the current resource utilization status: *heavy resource* utilization, indicating that a node does not have the ability to execute data-driven tasks locally; *normal resource* utilization, indicating that a node can execute locally a task depending on the percentage of data overlapping and task length. Finally, *light resource* utilization, where a data-driven task has a high probability to be executed locally. These decisions are defined according to specific threshold values. The EC resources utilization is represented as $\mathcal{Z} = \{z_1, z_2, z_3, \ldots, z_n\}$, while ϑ_1 and ϑ_2 denote threshold values. ϑ_1 points to heavy resources utilization, while ϑ_2 refers to medium resource utilization. A task T_k could be executed on $n_k \in \mathcal{N}_i$ only if n_k has enough resource for this execution. To reduce the percentage of failed tasks caused by resource utilization, the leader node n_i will request resource utilization from each node $n_j \in \mathcal{N}_i$. When n_i gets utilization information from neighboring $n_j \in \mathcal{N}_i$, it classifies it as:

$$\begin{aligned} \text{IF } (z_i \geq \vartheta_1) \rightarrow &\quad \text{Heavy Utilization} \\ \text{IF } (\vartheta_1 < z_i \leq \vartheta_2) \rightarrow &\quad \text{Normal Utilization} \\ \text{IF } (z_i < \vartheta_2) \rightarrow &\quad \text{Light Utilization} \end{aligned} \tag{3}$$

This rule states that if the state of z_k is low, that means the node n_k has a high resource availability, it can execute a data-driven task T_k even if the data overlapping percentage u_k is not high. Secondly, if the state of z_k is classified as normal, the node n_k can execute a data-driven task T_k under certain constraints, such as the percentage of data overlapping should be high and task sensitivity not very high. Finally, if the state of z_k is heavy utilization, it means the node n_k cannot execute a data-driven task T_k locally.

C. Delay Sensitivity

Delay sensitivity reflects data-driven task failure tolerance. The task sensitivity varies across tasks, e.g., there are urgent tasks related to healthcare and people safety requiring ultra-low latency [1], while some others are not urgent tolerating a delay for a few seconds or even minutes. To avoid WAN communication delays, important tasks should be given higher priority and executed on EC nodes. We introduce three levels of delay sensitivity to distinguish between urgent and non-urgent tasks: high, medium, and low sensitivity. Each incoming task is assigned to a specific level according to its urgency (e.g., forest fire propagation prediction tasks are highly sensitive, while online game tasks are of low sensitivity). Such classification helps our mechanism to assign the data-driven tasks to the appropriate EC nodes. We notate tasks' sensitivities in $\mathcal{S} = \{s_1, s_2, s_3, \ldots, s_n\}$, with $s_k \in [0, 1]$ based on [1] and [15]. If s_k value for task T_k is close to 1, it indicates high sensitivity, while a value around 0.5 or close to 0 indicates medium and low sensitivity, respectively. When leader n_i receives a set of data-driven tasks, it classifies their sensitivities to specific levels w.r.t thresholds Ψ_1 and Ψ_2:

$$\begin{aligned} \text{IF } s_i \geq \Psi_1 \rightarrow &\quad \text{High Sensitivity} \\ \text{IF } \Psi_1 \geq s_i \geq \Psi_2 \rightarrow &\quad \text{Medium Sensitivity} \\ \text{IF } s_i < \Psi_2 \rightarrow &\quad \text{Low Sensitivity} \end{aligned} \tag{4}$$

The rule states that, if task T_k has a sensitivity value greater than the first threshold, that means this task is particularly sensitive to the delay. As a result, n_i will point to this task as highly sensitive for delay (urgent) in order to take this into account when it makes the offloading decisions. On the other side, if the sensitivity value of a task T_k falls between the first and second thresholds, n_i will treat this task as a normal sensitivity. Finally, if task T_k has a sensitivity value outside the range of the first and the second thresholds, n_i will deal with this task as low sensitive for delay (un-urgent).

VI. Task Management Reasoning

In this section, we go over the proposed reasoning mechanism in greater details. The proposed mechanism takes into consideration the above mentioned parameters to proceed with task offloading decisions by balancing between nodes' capability and nodes' data availability to determine the appropriate location of the offloading data-driven tasks [1]. In such reasoning, two steps are introduced: acquiring tasks information and adopting FL inference.

The proposed mechanism runs on a specific node n_i which plays the role of the 'leader' in the neighborhood $\mathcal{N}_i$. 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 widely adopted leader election mechanisms. We do not elaborate on these mechanisms, since it is beyond the scope of this paper. In the remainder, for simplicity of notation, we assume that node n_i is assigned with this leadership role to execute the FL inference engine, where all neighboring nodes $n_j \in \mathcal{N}_i$ directly communicate with their leader n_i. The main goal for the first step is matching between tasks and corresponding nodes. That means assigning task T_k to node n_k that gives the highest data overlapping, lowest resource utilization and a high possibility to execute high sensitive data-overlapping tasks. However, this step determines n_k as the better choice for T_k compared to the rest of nodes $n_j \in \mathcal{N}_i$. Hence, we need accurately to measure the probability of offloading for each task, In order to decide whether task t_k should execute locally on n_i, offload to node n_k (action a_{11}) or offload to the cloud (action a_{12}). FL inference is adopted to determine such probability per task.

A. Task Information Updating

The leader node n_i collaborates with its neighbors to update tasks information. For each task, a neighboring available and suitable node can be assigned based on the following reasoning. Firstly, leader n_i asks for the task context (u_k, z_k, s_k) for each task T_k from its neighbours $n_j \in \mathcal{N}_i$. The goal is to determine how much data accessing it requires in node n_j, percentage of resources utilization in node n_j and expected delay in node n_j. Once n_j receives the request from leader n_i, it sends over (u_k, z_k, s_k) for each task T_k according to its local data $\mathcal{D}_j$. When the leader n_i receives information from n_j, it then has two pieces of context: the main context according to its data D_i and a new one received from n_j. The leader n_i in turn, makes a comparison between its tasks information and neighbours tasks information. It then updates T_k tasks information from n_j based on the following rule:

$$\text{IF } (u_{k.j} > u_{k.i}) \text{ OR } (z_{k.j} > z_{k.i}) \text{ OR } (s_{k.j} > s_{k.i}) \text{ THEN } (u_{k.i}, z_{k.i}, s_{k.i}) \rightarrow (u_{k.j}, z_{k.j}, s_{k.j}) \quad (5)$$

The rule states that the task T_k's context $(u_{k.i}, z_{k.j}, s_{k.j})$ in leader node will be updated if the corresponding values from neighbouring n_j are greater; otherwise, the task's context will not be updated. The rationale behind updating tasks context is based on achieving lower r_k. Therefore, if $(u_k, z_{k.j}, s_{k.j})$ are not greater than the ones in leader node, that means r_k increases and this leads to increase the probability of offloading task to unsuitable node or to the cloud. In order to avoid such a situation, context will not be updated. This process will be repeated for all the tasks getting context form each neighboring node $n_j \in \mathcal{N}_i$ sequentially. Hence, with each received task information from the next node $n_{j+1} \in \mathcal{N}_i$, if the rule is fired, the leader's task context keeps updating. By the end of this process, leader n_i will have updated all the required information; according to the suitable nodes $n_j \in \mathcal{N}_i$.

B. Fuzzy Linguistic Modeling

FL inference has been adapted to handle the inherent uncertainty and approximation of these contextual factors in EC environments. FL is the most prevalent strategy for dealing with rapid change in uncertain systems [15]. Fuzzy inference relies on rules over *linguistic variables* that can model such type of uncertainty. The main advantage for FL inference performed locally on a node is its lightweight computational complexity while providing an explainable decision-making methodology [11][15]. This explainability is based on linguistic variables reflecting the uncertainty derived from the values of u_k, z_k, and s_k. The data (fuzzy variable) is associated with three linguistic fuzzy values $\{\text{High}, \text{Medium}, \text{Low}\}$ reflecting a high, medium, and low value of data overlapping derived from the task's query data subspace over node's data space. Similarly, the resources utilization (fuzzy variable) is associated with the linguistic values $\{\text{Heavy}, \text{Normal}, \text{Light}\}$ reflecting a high, medium, and low value resources utilization. Finally, the delay sensitivity indicator o_k (as a fuzzy variable) takes three linguistic values: $\{\text{High}, \text{Medium}, \text{Low}\}$ reflecting the level of urgent for a data-driven task T_k. 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 the membership function $\mu_u^H(x) \in [0, 1]$. For instance, if the data overlapping $u_k = 0.7$ for task T_k, then this can possibly be considered as a High data overlapping with possibility $\mu_u^H(0.7) = 0.88$. Then, membership functions have been defined for all the parameters to be used in fuzzification and defuzzification steps. In this context, we have three membership function sets, and each set has different function according to the related linguistic variables [15]. There are different membership functions forms that can be adopted for fuzzy based reasoning such as trapezoidal, piecewise linear, singleton, triangular, and Gaussian [4]. We chose the triangular form to represent the membership functions, which is considered as the most common form according to [11][15]. Our membership functions are shown in Fig 2.

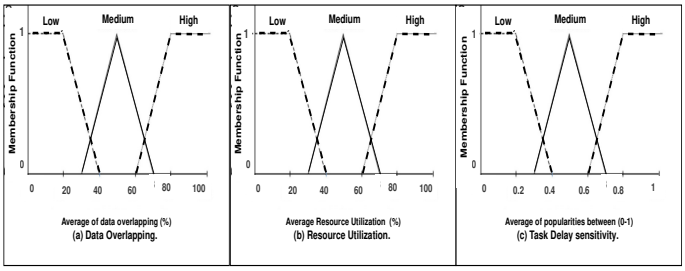

Fig. 2: Membership functions of our fuzzy variables.

The values employed in the membership functions, on the other hand, have a substantial impact on FL performance. Therefore, defining these values is considered to be a critical task. Therefore, similar to several previous studies about fuzzy decision mechanisms in the literature, the degree of membership values for fuzzy variables are defined empirically [15].

C. Fuzzy Logic-based Reasoning

The inputs and the outputs of the FL are variable from the natural language as we have mentioned above (e.g., High, Medium, Low). Given the set of membership functions, we introduce a novel FL reasoning engine that makes the decision of task execution locally (actions a_0), offloading to another node $n_j \in \mathcal{N}_i$ (action a_{11}) or offloading to the cloud (action a_{12}). *The offloading probability* for a task T_k is determined by the output of the FL inference system given the input u_k, z_k, and s_k, as will be elaborated later in this section. The FL reasoning process on n_i for each task T_k goes through the following steps: The first step is fuzzification of the inputs (u_k, z_k, s_k) into their fuzzy linguistic terms via the membership functions. Since it takes all these factors as

numerical values (crisp values), then assign fuzzy class for each fuzzy variable as follows [1][11].

$$\begin{aligned} F_{u_k}(x) &= \{\mu^L_{u_k}(x), \mu^M_{u_k}(x), \mu^H_{u_k}(x)\} \\ F_{z_k}(x) &= \{\mu^L_{z_k}(x), \mu^N_{z_k}(x), \mu^H_{z_k}(x)\} \\ F_{s_k}(x) &= \{\mu^L_{s_k}(x), \mu^M_{s_k}(x), \mu^H_{s_k}(x)\} \end{aligned} \tag{6}$$

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 [15]. The antecedent part ('IF' part) is a set of logical conjunctions over the fuzzy linguistic variables. The consequent part ('THEN' part) of the FIR is a fuzzy term from the set of linguistic terms {Low, Medium, High} that expresses the offloading probability r_k. The generic format of the FIR statements used in our engine is as follows:

$$\begin{aligned} \text{IF } u_k \text{ IS } X_1 \text{ AND } z_k \text{ IS } X_2 \text{ AND } s_k \text{ IS } X_3 \\ \text{THEN } r_k \text{ IS } X_4 \end{aligned} \tag{7}$$

where the linguistic terms $X_1, X_3, X_4 \in$ {Low, Medium, High} and $X_2 \in$ {Heavy, Normal, Light}. For instance, the following FIR:

IF u_k IS HIGH AND z_k IS LIGHT AND s_k IS HIGH
THEN r_k IS LOW.

This rule indicates that the probability of offloading T_k is low, which means action a_0 is preferred more than action a_1, due to the fact that the data required by this tasks can be fully available to node n_i (high degree of overlapping). Also, n_i has a low resource utilization and low expected execution delay. Hence, in this case, T_k can be locally executed on node n_i and not being offloaded (i.e., low offloading probability). Our engine requires 27 FIRs in the fuzzy knowledge base in order to cover the whole decision space; there are $3^3 = 27$ membership functions involved in the three fuzzy variables: data overlapping, resource utilization, and task delay sensitivity. Some FIRs of the engine are provided in Table I, which reflect the reasoning behind the decision on the actions a_0, a_{11} or a_{12} represented via the offloading probability.

TABLE I: FIRs: inputs and expected output (sample).

FIR	u_k	z_k	s_k	r_k
1	Low	Light	Low	Medium
2	Low	Light	Medium	Medium
3	Low	Light	High	High
4	Low	Normal	Low	Medium
5	Low	Normal	Medium	Medium
6	Low	Normal	High	Medium

The last step is the defuzzification of all the offloading probability values of the activated FIRs [11], [15], 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 FIRs. We adopt the centroid defuzzifier, which is considered as the most common operator while the defuzzified value represents probability aligned with the notion of r_k:

$$r_k = \frac{\int_{x\in[0,1]} x\mu^{\nu}_{r_k}(x)}{\int_{x\in[0,1]} \mu^{\nu}_{r_k}(x)}, \tag{8}$$

where ν is a {Low, Medium, High} linguistic terms of the offloading probability. The defuzzified offloading probability r_k ranges between 0% and 100%. To transform this probability to a decision, we define three decision thresholds: the first threshold is 30%, if r_k for task T_k in n_j is less than or equal 30%, the decision is to locally execute this task (action a_0) or offload it directly to the suitable node n_j (action a_{11}). Since having r_k less than or equal 30% with a task T_k in node n_j means this task has a high data overlapping with this node, and this node has enough resources to execute this task according to the time consecrates. Second threshold is 70%, if r_k for a task T_k is less than or equal 70%, the decision is offload to n_j if it is available or to the cloud actions, i.e., a_{11} or a_{12}. That means, if a task T_k has almost high or medium data overlapping, normal resource utilization and has medium sensitivity to delay with n_j. Thus, it could offload for this node if it is available. Otherwise, to the cloud. The third threshold is higher than 70%, if r_k for a task T_k in n_i is greater than 70%, the decision is to offload to the cloud. Since this task does not has a high data overlapping with any node $n_j \in \mathcal{N}_i$, as shown in Fig.3.

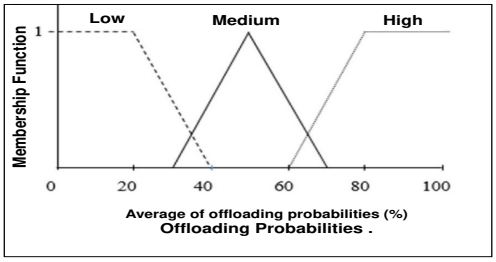

Fig. 3: The probability of offloading.

That means offloading more data, this could affect negatively resource utilization and increase other tasks' delays. In this case, in order to avoid these issues, these types of tasks offloaded to the cloud.

VII. Experimental Evaluation

We deal with data overlapping over analytics queries on real datasets. EdgeCloudSim simulator has been used to measure the impact of our mechanism on upload/download tasks, resource utilization, and task delay sensitivity.

A. Data Overlapping Experiment

We experiment with data overlapping over real datasets that have been collected by four Unmanned Surface Vehicles (USVs) acting as nodes n_i to collect data from sensors in a coastal area[1]. Each USV node n_i has a neighborhood

[1]https://archive.ics.uci.edu/ml/datasets/GNFUV+Unmanned+Surface+Vehicles+Sensor+Data+Set+2.

$\mathcal{N}_i \subset \mathcal{N}$ of directly communicating nodes $n_j \in \mathcal{N}_i$. Moreover, node n_i communicates with end-users/applications to collect data and store them locally in D_i for predictive analytics tasks. Each USV's dataset contains 2-dimensional points with features: sea surface temperature and humidity, i.e., $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^d$. There is one node $n_i \in \mathcal{N}$ acting as leader that receives analytic tasks T_k and decides either to locally execute T_k (action a_0) or offload (action a_1). The leader n_i receives requests for ten predictive analytic tasks including regression, classification, outliers detection, missing values substitution, novelty identification, and clustering, and it needs to obtain the u_k, z_k, and s_k for each T_k. Regarding the task's data overlapping u_k, we have defined for each local dataset $\mathcal{D}_i$, the feature boundaries $\max$ & $\min$ values: $\mathbf{D}_i = [x_1^{\min}, x_1^{\max}, x_2^{\min}, x_2^{\max}]$. Then, we generated uniformly at random 1,000 tasks queries (100 queries per task) $\mathbf{q}_k$ such that $\mathbf{q}_k = [q_1^{\min}, q_1^{\max}, \ldots, q_d^{\min}, q_d^{\max}]$ for each T_k to obtain the data subspace needed. 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 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], outlier o_k either 0 or 1, while the percentages of data overlapping u_k are between [0%, 100%]. All of these are inputs to the FL system, while the probability of offloading r_k is the output in [0%, 100%].

TABLE II: Query generation & Data overlapping (sample).

$\mathbf{q_1^{\min}}$	$\mathbf{q_1^{\max}}$	$\mathbf{q_d^{\min}}$	$\mathbf{q_d^{\max}}$	**Points including**	**Percentage**
19	32	49	57	130/899	14.46%
19	29	44	46	164/899	18.24%
26	28	43	58	75/899	8.3%

B. Experimental Setup & Context

To build the considered scenarios, we used CloudSim Plus to evaluate our mechanism's performance. We compared our results to the model proposed in [16]. The comparison was made in terms of bandwidth, resource utilization, and task execution time. Two types of parameters have been used: data-driven task characteristics and edge/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 EC nodes, while others are beyond EC node's capabilities and should be offloaded to the cloud. To simulate real-world 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 III contains tasks information chosen based on [15]. 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, (50000MB, 100MB) respectively denote the size of uploaded data (humidity, temperature, wind, etc.) that will be used to build a ML model, and download the model that the application will receive as a result of data collecting and training in EC/cloud computing. Task length (number of Million Instructions (MI)) on the other hand, determines the required CPU resource to complete a data-driven task. Other simulation parameters are listed in Table IV. We have considered ten tasks arriving at n_i, with specific features, which are task length, 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 [15].

TABLE III: Application Types used in the simulation.

Task	**Application**	**Task Length**	**Upload/Download data**
T1	Deep learning	10000	50000/100
T2	Traffic jam prediction	20000	200000/300
T3	Air pollution prediction	15000	200000/400
T4	Healthcare diagnosis	30000	80000/100
T5	Weather prediction	8500	50000/50
T6	Compute-intensive task	20000	300000/500
T7	Fraud detection	18000	300000/250
T8	Virtual assistants	25000	20000/50
T9	Alerting And Monitoring	14000	100000/300
T10	Social Media Analysis	21000	60000/80

TABLE IV: Simulation Parameters.

Parameters	EC	Cloud
Bandwidth	WAN 500MB/sec	LAN 10GB/sec
Number of VM	2	8
Number of cores	2	8
VM CPU speed	10 MB	100 MB
HOST MIPS	1000	10000

C. Comparative Assessment & Results

We compare the effectiveness of our mechanism against two alternative mechanisms over the same tasks simulation conditions. The first one, cloud-based mechanism [16], where the EC nodes have been used to collect sensors data and send it to the cloud, to reduce sensors' energy consumption that would happen if data has been sent directly to the cloud. The second mechanism, EC-based mechanism, has been suggested in many studies, such as [1], [15] where the tasks are sent to the EC 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, whereas the uploading speed in the cloud-based model is between 28 to 60 minutes. While the uploading speed in EC mechanisms, which has not considered data overlapping, is almost double the speed we obtained with our mechanism (see Fig. 4).

In terms of execution time, we have considered data offloading time beside the main execution time, because, in the data-driven tasks, data are considered to be an integral part of the task execution. Fig. 5 shows that the execution time is extremely minimized compered to the cloud-based model.

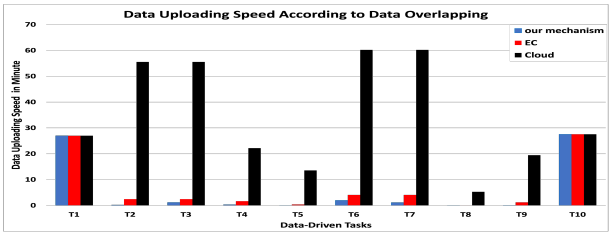

Fig. 4: Data uploading speed.

Also, we can observe that the bandwidth has been reduced as well.

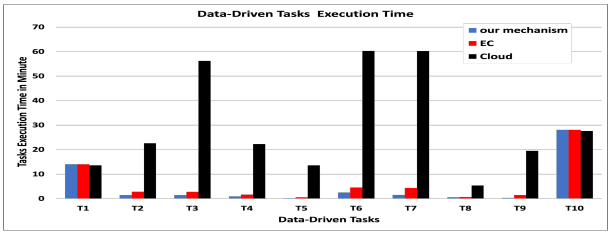

Fig. 5: Data-Driven tasks execution Time.

The results of cloud (WAN) and EC (MAN) bandwidth measurements are shown in Fig. 6. The blue bar represents the bandwidth usage percent according to our mechanism, which is considered to be very low compared to the other mechanisms. The red bar depicts bandwidth utilization according to the EC-based mechanism, which is nearly double that of our mechanism. Meanwhile, the black bar shows the bandwidth usage in order to execute these ten tasks on the cloud, which is very high usage compared to ours and EC-based (see Fig. 6). In terms of resource use, the cloud-based

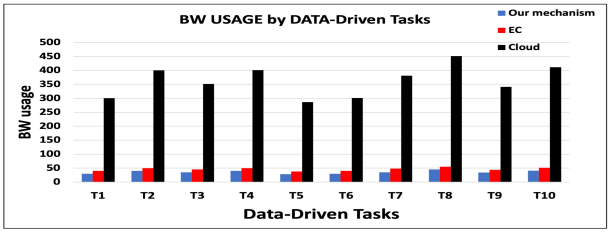

Fig. 6: Data-Driven Bandwidth Usage.

mechanism outperforms both our and EC-based mechanisms because it has unlimited resources. (Fig. 7).

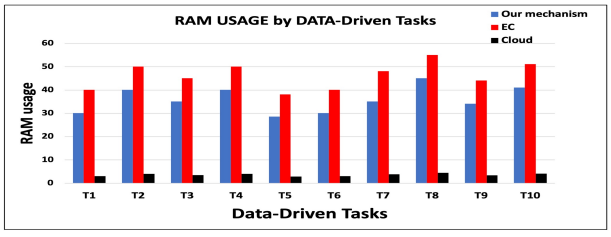

Fig. 7: Data-Driven Tasks RAM Usage.

VIII. Conclusions

In this work, we introduced a mechanism for data-driven analytics task execution in EC environment with the objective of exploiting their resources efficiently. In particular, the core of this mechanism focuses on three factors to make the execution decision for each task: data overlapping, EC resource utilization and task sensitivity. Such factors are inputs to a FL system to derive the probability of task offloading. Our mechanism significantly outperforms other benchmarks in terms of reducing uploading data size, execution time and bandwidth and RAM usage. We plan to improve our mechanism by taking into account EC mobility.

References

[1] Jaber Almutairi and Mohammad Aldossary. A novel approach for iot tasks offloading in edge-cloud environments. *J. Cloud Computing*, 10(1):1–19, 2021.

[2] Ishan Budhiraja, Neeraj Kumar, Sudhanshu Tyagi, and Sudeep Tanwar. Energy consumption minimization scheme for noma-based mobile edge computation networks underlaying uav. *IEEE Systems*, 2021.

[3] Qiong Chen, Zimu Zheng, Chuang Hu, Dan Wang, and Fangming Liu. On-edge multi-task transfer learning: Model and practice with data-driven task allocation. *IEEE TPDS*, 31(6):1357–1371, 2019.

[4] Zhixiong Chen, Nan Xiao, and Dongsheng Han. Multilevel task offloading and resource optimization of edge computing networks considering uav relay and green energy. *Applied sciences*, 10(7):2592, 2020.

[5] Jun Cheng and Dejun Guan. Research on task-offloading decision mechanism in mobile edge computing-based internet of vehicle. *J. Wireless Communications and Networking*, 2021(1):1–14, 2021.

[6] Kuntao Cui, Bin Lin, Wenli Sun, and Wenqiang Sun. Learning-based task offloading for marine fog-cloud computing networks of usv cluster. *Electronics*, 8(11):1287, 2019.

[7] Kostas Kolomvatsos and Christos Anagnostopoulos. A deep learning model for demand-driven, proactive tasks management in pervasive computing. *IoT*, 1(2):240–258, 2020.

[8] Fanhui Kong, Jian Li, Bin Jiang, Tianyuan Zhang, and Houbing Song. Big data-driven machine learning-enabled traffic flow prediction. *Trans. Emerging Telecomm. Technologies*, 30(9), 2019.

[9] Shuyang Li, Xiaohui Hu, and Yongwen Du. Deep reinforcement learning for computation offloading and resource allocation in unmanned-aerial-vehicle assisted edge computing. *Sensors*, 21(19):6499, 2021.

[10] Mithun Mukherjee, Suman Kumar, Constandinos X Mavromoustakis, George Mastorakis, Rakesh Matam, Vikas Kumar, and Qi Zhang. Latency-driven parallel task data offloading in fog computing networks for industrial applications. *IEEE TII*, 16(9):6050–6058, 2019.

[11] VanDung Nguyen, Tran Trong Khanh, Tri DT Nguyen, Choong Seon Hong, and Eui-Nam Huh. Flexible computation offloading in a fuzzy-based mobile edge orchestrator for iot applications. *Journal of Cloud Computing*, 9(1):1–18, 2020.

[12] VanDung Nguyen, Tran Trong Khanh, Nguyen H Tran, Eui-Nam Huh, and Choong Seon Hong. Joint offloading and ieee 802.11 p-based contention control in vehicular edge computing. *IEEE Wireless Communications Letters*, 9(7):1014–1018, 2020.

[13] Zhaolong Ning, Peiran Dong, Xiaojie Wang, Joel JPC Rodrigues, and Feng Xia. Deep reinforcement learning for vehicular edge computing: An intelligent offloading system. *ACM TIST*, 10(6):1–24, 2019.

[14] Fatima Samea, Farooque Azam, Muhammad Rashid, Muhammad Waseem Anwar, Wasi Haider Butt, and Abdul Wahab Muzaffar. A model-driven framework for data-driven applications in serverless cloud computing. *Plos one*, 15(8):e0237317, 2020.

[15] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Fuzzy workload orchestration for edge computing. *IEEE Transactions on Network and Service Management*, 16(2):769–782, 2019.

[16] Pengfei Wang, Ruiyun Yu, Ningwei Gao, Chi Lin, and Yonghe Liu. Task-driven data offloading for fog-enabled urban iot services. *IEEE Internet of Things Journal*, 8(9):7562–7574, 2020.