In-Network Placement of Delay-constrained Computing Tasks in a Softwarized Intelligent Edge

Gianmarco Lia, Marica Amadeo, Giuseppe Ruggeri, Claudia Campolo, Antonella Molinaro, Valeria Loscrì

In-Network Placement of Delay-constrained Computing Tasks in a Softwarized Intelligent Edge

Gianmarco Lia[1,2], Marica Amadeo[1,2], Giuseppe Ruggeri[1,2],
Claudia Campolo[1,2], Antonella Molinaro[1,2,3], Valeria Loscrì[4]
[1]University Mediterranea of Reggio Calabria, Italy
Email: {name.surname}@unirc.it
[2]Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Italy
[3]Laboratoire des Signaux et Systémes, CentraleSupélec, Université Paris-Saclay, France
[4] Inria Lille–Nord Europe, France
Email: valeria.loscri@inria.fr

Abstract

Future sixth-generation (6G) networks will rely on the synergies of edge computing and machine learning (ML) to build an intelligent edge, where communication and computing resources will be jointly orchestrated. In this work, we leverage ML algorithms to judiciously orchestrate the placement of delay-constrained computing tasks in a softwarized edge domain. A set of popular supervised learning algorithms, i.e., Decision Tree (DT), Bagged Trees (BTs), Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), have been leveraged to this purpose. They are trained off-line through the results of an optimization problem targeting the minimization of the edge network resource usage while respecting the tasks' delay constraints. Extensive simulation results are reported to showcase the performance of the considered techniques in terms of model accuracy, complexity and network-related metrics, e.g., amount of exchanged data in the edge domain. Among the compared techniques, DT and MLP are shown to be the most efficient solutions in terms of algorithm execution time, by achieving almost the same performance.

Keywords: Task placement, Edge Computing, Machine Learning, In-network computing, 6G, Intelligent Edge, Edge AI

1. Introduction

Edge computing enables data processing at the periphery of the network, close to where data themselves are generated [1]. Compared to the traditional cloud-based solutions, where processing is executed in remote data centers, executing tasks at the edge offers benefits in terms of latency and network traffic reduction for a large variety of interactive and delay-sensitive applications, like on-line gaming, surveillance, and Augmented Reality (AR).

In multi-user edge domains, composed of multiple edge nodes, equipped with heterogeneous and limited (compared to the cloud) communication and computing capabilities, it is hard to deploy practical task placement strategies, based on which specific data processing tasks are allocated to specific edge nodes (playing as task *executors*). This is even more true in the presence of challenging sixth generation (6G) applications, with different and much stricter requirements [2], like EXtended Reality (XR) and autonomous driving.

Regardless of the specifically targeted optimization criteria, e.g., minimization of energy consumption, network bandwidth, latency [3], [4], traditional task placement strategies rely on the solution of computationally expensive NP-hard problems [5]. Near-optimal sophisticated heuristics can be conceived to achieve a solution in a feasible amount of time, but they are not easy to derive.

The recently proposed concept of *intelligent edge* [6], deemed a key 6G enabler, aims at incorporating machine learning (ML) techniques into the edge for the dynamic, adaptive and efficient orchestration of communication and computing resources [7]. Exploiting ML allows to replace a complex mathematical modelling of the system to derive mathematically tractable heuristics with a data-based understanding of the system [5].

As surveyed in [8], supervised learning-based and unsupervised learning-based solutions can be devised for the sake of deciding whether and how to offload computing tasks. Among them, supervised learning techniques are effective to provide near-optimal heuristic-like offloading decisions in a scalable and efficient manner [9]. This is the reason why in this paper we focus on supervised learning techniques.

Currently, the majority of works on intelligent edge solutions focus on task offloading in the presence either of a single edge server [10] or of a restricted number of purpose-built edge servers, e.g., co-located with the base stations of a Radio Access Network [8]. There is a lack of comprehensive studies that well assess the performance of ML algorithms in 6G edge domains with multiple network nodes characterized by heterogeneous communication and computing capabilities, which could be selected as task executors. Indeed, according to recent initiatives like the Internet Research Task Force (IRTF) Computing in the Network (COIN) Research Group [11], 6G edge nodes acting as task executors are not only purpose-built servers. They may encompass potentially any network edge node augmented with computing resources, such as the ones composing a campus network, the backhaul segment of a mobile network, or a metropolitan area network [12].

In this paper, we focus on a distributed network edge domain with multiple heterogeneous nodes playing the role of candidate task executors, and apply *supervised learning techniques to efficiently solve an optimal computing task placement problem*. This latter has been preliminarily formulated in [13] as an Integer Linear Programming (ILP) problem that targets *the minimization of the network edge resource usage* by reducing the amount of data exchanged within the edge domain, while ensuring that *the delay constraints of each computing task are met*.

The optimal solution computed offline is leveraged to train supervised learning algorithms. The latter are feeded by the information about the task requests and edge domain state, and target to solve a multi-class classification problem, providing the task placement decision as output.

In so doing, optimal solutions do not require to be computed online, providing a less complex and quicker placement decision making, at the expenses of heavy training procedures to be performed offline.

To deploy the conceived solution, we consider a softwarized network edge infrastructure, based on Software-Defined Networking (SDN) [14]. Indeed, the centralized view of the domain, available at the SDN controller, facilitates the orchestration of resources available at the network edge nodes [12] and makes easier the application of ML techniques [9].

More specifically, the paper significantly extends the work in [13] by providing the following main original contributions:

- A variety of popular supervised learning algorithms, including Decision Tree (DT), Bagged Trees (BTs), Multi Layer Perceptron (MLP), and Support Vector Machine (SVM), are designed and implemented aimed at identifying the near-optimal placement of delay-constrained computing tasks in a network edge domain. The goal is to minimize the edge domain resource usage while satisfying the tasks' delay constraints.

- An extensive evaluation campaign is performed to assess the efficiency (in terms of algorithm execution time) and accuracy performance of the considered ML techniques, when compared to the formulated ILP's optimal solutions.

- The considered ML techniques achieving the lowest algorithm computational time complexity are then compared to assess their effectiveness and efficiency in placing the tasks, while satisfying their deadline constraints. In addition to the network resource usage metric, expressed in terms of data exchanged at the edge, several task-related metrics, like the average computation time and the task deadline missing probability, are assessed under a wide variety of settings (e.g., rate of task requests, task delay constraints) to validate their ability to approximate the optimal solution.

To the best of our knowledge, this is the first comprehensive study comparing different ML techniques, which are typically evaluated *individually* in the task offloading literature [5], [8], [15], [16], to specifically solve the formulated task placement problem in a softwarized network edge domain made of several heterogeneous computing nodes. The assessment of their performance is provided in terms of execution times, as well as in terms of traditional ML-related metrics, like accuracy and precision, and in terms of meaningful network- and task- related metrics to assess their suitability to tackle the problem at hand

The remainder of the paper is organized as follows. Section 2 scans the closest related literature and presents the motivations of our work. The system model and optimization problem are presented in Section 3. The proposed ML-based framework and the dataset generation are respectively discussed in

Section 4 and Section 5. The performance assessment is reported in Section 6, before conclusive remarks and hints on future works in Section 7.

2. Background

Several works in the recent literature have investigated the task placement problem in edge computing environments [1], [17]. Some of them focus on single-user edge systems, where the decision to be taken is on whether a particular task should be (partially) offloaded to the edge for being executed or instead computed locally on the end-device. Others target multi-user edge systems encompassing multiple clients that can be served by a single edge server. When focusing on 5G and beyond environments, however, more complex scenarios are considered, where real-time compute-intensive tasks, issued by different clients, can be potentially executed by multiple edge servers. In this context, a variety of challenges need to be addressed, including *(i)* managing the coexistence of distributed edge computing and centralized cloud systems [18]; *(ii)* determining the best executor of each computing task [17]; *(iii)* optimizing the communication and computational resource allocation at the edge in the presence of mobile users [19], [20].
In particular, in optimal task placement problems, metrics like the latency, the energy consumption, the bandwidth usage, are typically considered, either disjointedly or jointly [3], [4], [21].

Besides being executed by purpose-built servers, tasks can be also placed into network edge nodes augmented with computing capabilities [12], [20]. Treating network nodes as computing equivalents is the main pillar of the recent *in-network computing* paradigm[1] [13], [22]. In such a context, characterized by the presence of several candidate executors with limited and heterogeneous computing capabilities and differently chained to the requester nodes, the task placement decisions get dramatically complicated.

Incorrect decisions are unable to meet the user requirements and can degrade the efficiency of the system.

2.1. ML-based task placement

Instead of devising mathematically tractable heuristics or decomposing problems with high levels of dimensionality in smaller sub-problems to reduce their complexity, ML approaches can be applied [23] that provide near-optimal heuristic-like decisions [8]. According to [6], by introducing edge intelligence into task placement and resource allocation mechanisms, the performance can be optimized even in highly dynamic scenarios with new users joining continuously.

The work in [24] focuses on the allocation of network resources to provide Service Function Chains (SFCs), i.e., the so-called Virtual Network Function Placement and Chaining (VNF-PC) problem. After proposing an ILP model,

[1]https://datatracker.ietf.org/rg/coinrg/about/

the authors develop a hybrid strategy combining the optimal approach with (unsupervised) ML to find the adequate network resources while decreasing the resolution time. In particular, the k-Means algorithm is applied by considering the request history and the geographic features of the edge servers. Results show that the hybrid strategy minimizes the runtime by up to 75% compared to the ILP method, without degrading the acceptance rate and the provider's profit. However, the k-Means algorithm is not efficient in the presence of highly dynamic scenarios, where clusters (whose number must be specified in advance) are potentially of varying sizes and density [6].

In parallel, supervised learning approaches like DT and SVM provide near-optimal heuristic-like offloading decisions in a scalable, flexible and computation-efficient manner [9] [25].

For instance, a popular DT algorithm, the Classification and Regression Tree (CART), is considered in [15] to select the best fog devices to which tasks should be offloaded. The decision, however, does not consider the maximum computation delay in processing the offloaded tasks. A CART algorithm is also leveraged in [5] for the sake of predicting the best placement of VNFs. An SVM model is applied in [26] by converting the decision of offloading subtasks to edge servers from vehicles into classification problems. An ML-based workload orchestration approach for vehicular environment is also proposed in [16]. There, an MLP model was chosen, after testing other techniques, as an extremely flexible classifier model which predicts whether the results of the offloading options are successful or not for each target device.

Another well-known category of supervised learning algorithms applied in edge intelligence contexts includes Convolutional Neural Networks (CNNs) [6]. For instance, the work in [27] focuses on automatic speech analysis (ASA) tasks to be processed with minimum delay in a cloud-edge environment, by considering a user tolerance limit. Despite their high accuracy, CNNs still require long training time and result more complex and computationally expensive than other supervised learning approaches [28], [29]. Also, CNN is mainly based on black box models and, therefore, has lower explainability compared to DT and SVM approaches [6]. Compared to CNN, MLP has a much simpler structure and enables easier hyperparameter tuning thus contributing to the model explainability [30]. This is why, in the following, we do not consider CNNs in our analysis and, instead, we focus on DT, SVM and MLP approaches.

2.2. SDN-based task placement

In parallel, to optimize the usage of communication and computing resources of an edge domain, SDN-based solutions have been also proposed in the literature [31], [32]. Indeed, the information available at the SDN control plane, including the status of computing resources and communication links, as well as the resource requirements from users, can be leveraged to optimize the task offloading decisions.

In [33] a software-defined vehicular edge computing is proposed that employs edge computation to enable ultra-low latency vehicular services. In [20], SDN is leveraged to monitor the users' mobility over time. Based on this information, a centralized Multi-access Edge Orchestrator applies a Dynamic Programming approach to optimize the anticipatory allocation of communication and computational resources at the network edge, while not exceeding the maximum latency tolerated by the users.

In our previous work in [12], we considered a distributed edge domain managed by an SDN controller and designed a placement strategy for minimizing the task execution latency, without however targeting the reduction of the network resource usage. With focus on delay-constrained task placement in a softwarized edge domain, in [13], instead, we targeted *the minimization of the network resources usage* by reducing the amount of data exchanged within the edge domain. There, the problem has been only formulated as an Integer Linear Programming (ILP) model and it is referred to as *Delay-constrained Minimum data Edge task Placement* (DMEP).

In this paper, we make a step forward by considering supervised ML-based solutions to solve the same problem in a time efficient and scalable fashion.

The possibility of enforcing ML-based task placement through a centralized SDN controller [9], as we targeted in this work, has been investigated also in [34], [35], [36]. However, unlike the related literature, we consider a more challenging scenario, i.e., a softwarized edge domain with ubiquitous in-network computing capabilities [11]. Also, we perform a comprehensive assessment covering both complexity and accuracy performance of ML approaches and network/computing performance at the edge.

3. System model and problem formulation

3.1. Reference scenario and main assumptions

The reference scenario of our study is a network edge domain, supervised by an SDN controller, and composed of a set of network edge nodes equipped with computing capabilities, see Fig. 1. Like in vanilla SDN implementations, the controller and the edge nodes interact through a southbound interface [14]. In particular, the edge nodes transmit information about the status of their resources and requests for task execution, while the controller injects forwarding and task allocation rules.

Edge nodes are connected with each other through wired links and may act as executors of tasks requested by a variable number of clients. Focus is on *delay-constrained* computing tasks over certain input content(s), e.g., images, videos, environmental data. Therefore, the executor needs to collect the content(s) first, and then process them within a certain time limit.

A subset of edge nodes act as ingress nodes for clients and input data sources, while an egress node connects the edge domain with a remote cloud, where tasks can be offloaded when the edge domain is not able to process them, e.g., because the latency constraint cannot be met.

Input data producers can be: *(i)* the clients themselves that offload their own data; *(ii)* different nearby sources, e.g., cameras, sensors; *(iii)* a combination of both. Therefore, in principle, the input data for each computing task may be heterogeneous and retrieved by *multiple producers*.

Each edge node, $i \in N$, is provided with computing capabilities μ_i, expressed in terms of CPU cycles per seconds. For the sake of simplicity, we consider the remote cloud as a unique computing node c with large (potentially unlimited) computing capabilities.

Clients request the execution of a computing task, j, belonging to a catalog of size $|J|$. A computing task is atomic, i.e., it cannot be split in multiple sub-tasks and , similarly to other task models in the literature, e.g., [25], [37],

it is characterized by three parameters, $j = \{l_j, s_j, D_j^{max}\}$, where:

- l_j is the required amount of the computing resources, expressed in terms of CPU cycles, to execute the task j;
- s_j is the size of the input data for computing task j;
- D_j^{max} is the maximum delay constraint to execute the task j.

When task j needs to process distinct input data, the overall input data size, s_j, is equal to the sum of the size of K_j distinct input data, i.e., $s_j = \sum_{k_j=1}^{K_j} s_{k_j}$, being s_{k_j} the size of data k_j.

Task placement is orchestrated by the SDN controller as a *network application* (Task placement policy in Fig. 1). More specifically, an ingress node, receiving a new task request, forwards it to the controller. This latter chooses the best executor among the available edge nodes, if available, according to the policy described in Section 3.2 and implemented through ML techniques as described in Section 4. Thereafter, the executor retrieves the input data from the origin sources, performs the computation and returns the output to the client.

The forwarding paths to be followed by input data, from the origin sources to the task executor, and by the result of the computation, from the executor to the requesting node, are built by the SDN controller, according to the deployed Routing policy.

3.2. Optimization problem formulation

The target of the task placement problem is twofold: *(i)* limiting the network usage, expressed in terms of intra-domain traffic, *(ii)* ensuring that the maximum delay constraint of each computing task is met.

The formulation of the optimization problem targeting the above mentioned objectives has been proposed in our previous work [13]. For the sake of completeness, we report it in the following, while the main notation is summarized in Table 1.

Computation delay. The *total computation time* of task j at node i can be calculated as the sum of two contributions: *(i)* the *execution time*, i.e., the time needed to actually execute a task, and *(ii)* the *queuing time*, i.e., the time that a task has to wait before being executed because of the presence of other already running tasks. In our design, the execution time is calculated as the fraction between the amount of processing resources (CPU cycles) needed to

Table 1: Summary of the main notations in the optimization problem formulation.

Symbol	**Description**
J	catalog of tasks
N	set of SDN edge nodes
c	cloud node
$\widetilde{N}$	set of candidate executors $N \cup d$
i	generic edge node
μ_i	computing capabilities of node i (CPU cycles/s)
j	generic computing task
l_j	required computing resources to execute task j (CPU cycles)
s_j	overall size of input data of task j
k_j	generic input data of task j
K_j	number of input data of task j
s_{k_j}	size of the k_j-th input data of task j
D_j^{max}	maximum delay constraint for task j
λ_j	arrival rate of requests for task j
X_{ij}	binary decision variable: 1 if task j is executed by node i, 0 otherwise

execute the task (l_j) and the processing capability (CPU cycles/s) of the edge node which executes it (μ_i).

As commonly assumed in the related literature [4], [38] and justified by empirical studies [39], [40], we assume that the arrival rate of requests for task j at node i follows a Poisson distribution with parameter λ_j. The amount of computing resources per task, l_j, is exponentially distributed with average value $\bar{l}$. Therefore, each edge node can form an M/M/1 queuing model [41] to process its corresponding computing tasks [4]. Moreover, the service time of the queuing model is exponentially distributed with parameter $\frac{\bar{l}}{\mu_i}$.

The total request arrival rate at node i, that still follows a Poisson distribution, can be defined as:

$$\sum_{j \in J} X_{ij}\lambda_j, \tag{1}$$

where X_{ij} is the binary decision variable. It is equal to 1 if the task j is executed by edge node i and equal to 0 otherwise.
Thus, exploiting the queueing theory, the average computation time for a generic task at a node i can be derived as:

$$\overline{D_i} = \frac{1}{\dfrac{\mu_i}{\bar{l}} - \displaystyle\sum_{j \in J} X_{ij}\lambda_j}, \quad i \in \tilde{N}, \tag{2}$$

with $\tilde{N}$ being equal to $N \cup c$, i.e., the set of potential executors, including all edge nodes and the cloud. To keep the queue stable, the average arrival rate, λ_j, should be smaller than the average service rate (i.e., $\mu_i/\bar{l}$), as described by the following equation:

$$\mu_i/\bar{l} - \sum_{j \in J} X_{ij}\lambda_j > 0, \quad i \in \tilde{N}. \tag{3}$$

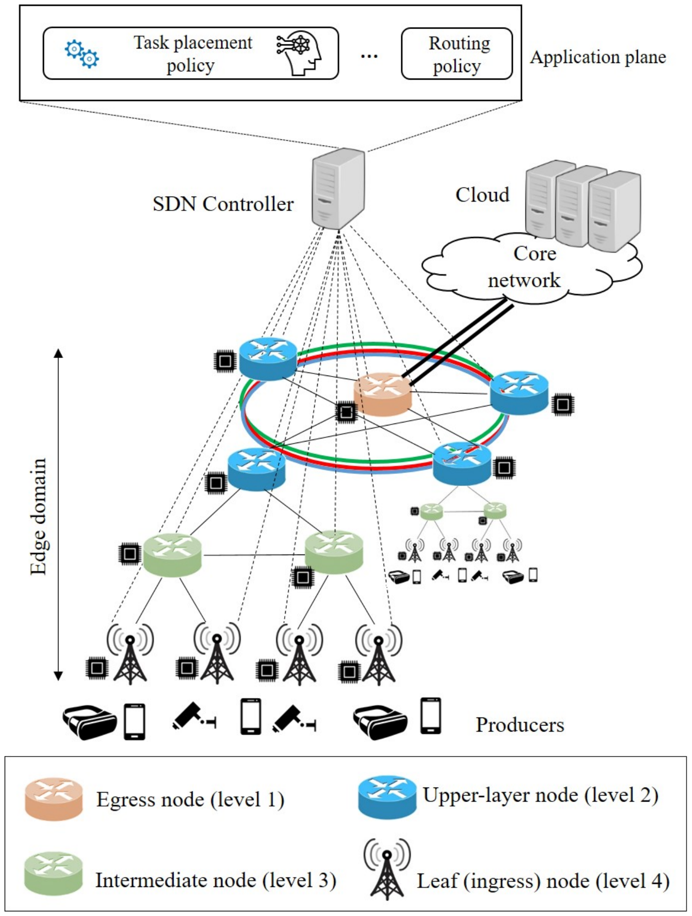


Figure 1: Reference Scenario.

Network usage. We denote by Ω_{i,k_j}^{min} the minimum number of hops between the candidate executor, i, with i=1, ..., $\tilde{N}$ and each content producer k_j feeding computing task j. We then define the *network usage* as a measure of the exchanged intra-domain traffic and derive it as:

$$NU = \lambda_j \sum_{k_j=1}^{K_j} s_{k_j}\Omega_{i,k_j}^{min}, \quad i \in \tilde{N}; j \in J. \tag{4}$$

The higher the hop number between the producer and the candidate executor, the higher the generated intra-domain traffic and, therefore, the network usage.

Optimization problem. Given the above mentioned parameters, the optimization problem can be formulated as follows:

$$\min \sum_{i\in\tilde{N}} \sum_{j\in J} X_{ij}\lambda_j \sum_{k_j=1}^{K_j} s_{k_j}\Omega_{i,k_j}^{min} \tag{5}$$

s.t.

$$\sum_{i\in\tilde{N}} X_{ij} = 1, \quad j \in J; \tag{6}$$

$$X_{ij}\overline{D_i} \leq X_{ij}D_j^{max}, \quad i \in \tilde{N};\ j \in J; \tag{7}$$

$$\frac{\mu_i}{\bar{l}} - \sum_{j\in J} X_{ij}\lambda_j > 0, \quad i \in \tilde{N}; \tag{8}$$

$$X_{ij} \in \{0,1\}, \quad i \in \tilde{N};\ j \in J. \tag{9}$$

Constraints in Eqs. (6) and (7), respectively, ensure that each task j is placed in one and only one edge node and is executed within its maximum tolerable delay, D_j^{max}. Considering Eq. (2), it is worth to note that constraint in Eq. (7) is described by nonlinear inequalities. Constraint (8) forces the average service rate of the edge node to be greater than the average task arrival rate, in order to keep the M/M/1 queue of the edge node stable. The constraint in Eq. (9) bounds the values of the optimization variable to be binary.

4. Supervised learning algorithms for task placement

The formulated ILP problem in Eq. (5) is equivalent to the Generalized Assignment Problem (GAP) [42], which is a well known NP-hard problem in the combinatorial optimization literature. In a realistic setting, its resolution through a standard optimization solver is not feasible due to the long time needed to find the optimal placement of the computing tasks.

Hence, a strategy is needed to efficiently solve the problem even if at the expenses of a sub-optimal solution. For this purpose, in alignment with the recent literature [8], we propose to leverage ML algorithms and, in particular, we choose to trace back the task placement problem to a *multi-class classification problem.* In this work, the scope of the ML-based technique is to predict the output, i.e., to select the task executor node, in response to a task request. The ML-based technique is implemented as a network application at the SDN controller. Indeed, thanks to its view of the edge domain (i.e., network topology, status of links, available computing resources of nodes) obtained by interacting with overseen nodes through the southbound interface [12], the SDN controller records various features that can be used as input to the classification models. Then, in response to a request for a computing task (with relevant demands) forwarded by an ingress node, the SDN controller leverages the ML-based technique to choose the edge node in charge of executing it.

More formally, we can define the input and output of our classification problem as follows.

Input. We associate to the m-th task request an input vector, $\overline{j_m}$. For a network with $|\tilde{N}|$ potential executors, each input vector is characterized by a vector of F features $\overline{j_m} = (j_{m,1}, j_{m,2}, ..., j_{m,F})$. After a long selection process, during which multiple choices have been tested, we set the number of features $F = 2 \cdot |\tilde{N}| + 1$, where:

- $j_{m,1}$: is the maximum computation delay of the task j_m.
- $(j_{m,i+1})$ with $i = [1, \ldots, |\tilde{N}|]$: represents the $|\tilde{N}|$ costs, expressed in terms of network usage, to execute the $m-th$ task on the $i-th$ node computed as follows:
$$j_{m,i+1} = \lambda_m \sum_{k_m=1}^{K_m} s_{k_m} \Omega_{i,k_m}^{min} \tag{10}$$
- $(j_{m,i+|\tilde{N}|+1})$ with $i = [1, \ldots, |\tilde{N}|]$: is a boolean vector of $|\tilde{N}|$ elements which capture the constraints of the stated problem. Specifically, if constraints expressed by Eqs. (7) and (8) will be satisfied for node i, the value of the i-th position of the vector will be 1, otherwise it will be 0.

Output: in response to the m-th input, each ML algorithm produces as output a value $y \in \tilde{N}$ that corresponds to the potential executor selected to serve the requested task. Once the executor is selected, the SDN controller notifies it about the computing task duty and builds the required routing paths for the task provisioning.

Several supervised ML techniques can be leveraged to address the formerly defined classification problems as extensively surveyed in [43]. After testing their performances, we selected a subset of them in alignment with related work on task offloading in the edge computing domain [5], [8], [10], [15], [16] to further assess their suitability to specifically tackle the problem at hand: *(i)* DT, *(ii)* BTs (ensemble method), *(iii)* MLP, *(iv)* SVM. Such techniques are shortly introduced in the following, whereas the main hyperparameters used for each of them are summarized in Table 2. The dataset generation, which is common to all of them, is instead described in the next Section.

4.1. Decision Tree

The DT method [44] exploits a binary tree-structured classifier to predict the response in front of an input. Each node of the tree corresponds to checking whether the value of a feature in the input vector satisfies a certain condition. If the condition is met, the method goes down to the *right* child, otherwise, it goes down to the *left* one. Each leaf node is associated with a response. It means that, given an input, if sequential decisions led to a particular leaf node, the response will be the class associated with that leaf node (the executor node for a given computing task in our case).

Table 2: Main hyperparameters and settings for the training of considered ML techniques in MATLAB®.

ML algorithm	**Setting**
Decision Tree	• Decision Tree algorithm: CART [44] • Maximum number of split: 500 • Split criterion: Maximum deviance reduction • Surrogate decision splits: Off • Optimizer: Bayesian Optimization [45] • Acquisition function: Expected improvement per second plus
Bagged Trees	• Ensemble method: Bag [46] • Learner type: Decision Tree • Maximum Number of splits: 488000 • Number of learners: 30 • Learner rate: 0.1 • Number of predictors to sample: all
MLP	• Number of features: 61 • Number of input layer neurons: 61 • Number of output layer neurons: 30 • Activation function: Softmax function • Loss function: Crossentropy • Optimizer: Scaled conjugate gradient [47] with: sigma=5x10^{-5}, lambda=5x10^{-7}, mu=5x10^{-3}
Linear, Quadratic, Cubic SVM	• Kernel function: linear, quadratic, cubic • Box constraint level (C): 1 • Kernel scale mode (γ) : Auto • Multiclass method: One-vs-One • Standardize Data: yes • Optimizer: Bayesian Optimization [45] • Acquisition function: Expected improvement per second plus

4.2. Bagged Trees (ensemble method)

BTs have been designed for improving the performance of a single DT. The basic idea is to consider multiple decision trees at the same time and to get the output by aggregating the output of each tree. To obtain multiple DTs, multiple training sub-sets are created by choosing random and repeatable samples from the original dataset. Each subset is then used to train a different DT [48].

4.3. Multi Layer Perceptron

MLP is a class of feed-forward Artificial Neural Network (ANN) [49] that consists of three or more layers of neurons: the input layer, the output layer and one or more hidden layers. Each neuron is characterized by a nonlinear activation function; the *softmax* activation function is commonly used in classification problems. When dealing with these kinds of problems, as in our study, the purpose of an MLP is to classify an object starting from a particular set of features which characterize the object. The number of features to be considered corresponds to the number of neurons in the input layer, while each neuron in the output layer corresponds to one of the possible classes an object may belong to. During the training phase the connections between the neurons are tuned so to respond to each input set of features by activating a single output neuron corresponding to the predicted class of the object characterized by those features.

4.4. Support Vector Machine

Given a set of objects characterized by F features, an SVM [50] classifies them by finding the *best hyperplane*, in the *F-dimensional* features space, that separates all data points of one class from those of another class. It is referred to as the *best hyperplane* the one which has the greatest distance between the two classes (or between each couple of classes if it deals with a multi-class classification problem). Support Vectors are the nearest samples of two different classes. They are used to calculate the distance between classes and the consequent hyperplane. In addition to perform linear classification, as formerly described, SVMs can efficiently perform a non-linear classification by mapping the original features to a higher-dimensional space by means of a non-linear kernel function. In our study, besides linear SVM, we considered SVM algorithms with quadratic and cubic kernel functions.

5. Dataset generation

The dataset used to carry out our experiment with each of the aforementioned supervised learning methods is generated by exploiting the optimal solution of the algorithm in Section 3, achieved through a standard optimization solver. In particular, the dataset generation was made by executing $10,000$ runs, which resulted in $487,997$ samples (task requests).

The generated database has been splitted into the training and validation sets according to the k-fold cross validation method, with $k = 10$ [51], [52]. According to this approach, the dataset is divided in 10-folds of equal size employing random sampling excluding repetition. Therefore, the models are trained 10 times. For each training session, a different 1 fold is used as validation set and the remaining ones as training sets. At the end of the process, the prediction error is estimated as the mean average of the 10 individual errors achieved during each training session. Thus, the above procedure allows to obtain a performance evaluation of the observed model unbiased by any specific partitioning of the dataset into training and validation subsets

Table 3: Main simulation settings.

Parameter	**Setting**
Processing capability (μ_i)	• Leaf nodes: $250 \cdot 10^6$ CPU cycles/s • Intermediate nodes: $500 \cdot 10^6$ CPU cycles/s • Upper-layer /egress nodes: 10^9 CPU cycles/s • Cloud: $10 \cdot 10^9$ CPU cycles/s [53]
Maximum computation delay (D_j^{max})	uniformly distributed in [10, 100] ms [54], [55], [56]
Average task computing workload ($\bar{l}$)	$10 \cdot 10^6$ CPU cycles [57]
Input data size (s_j)	10 MB [58], [59], [60]
Arrival rate of requests for task (λ_j)	10 requests/s [61]

6. Performance Evaluation

6.1. Main objectives

The conducted analysis aims to assess the performance of the proposed ML-based task placement strategies against the solution of the ILP problem (i.e., DMEP) achieved through a standard optimization solver. All the compared schemes have the objective to choose the best edge nodes in which the tasks have to be placed in order to minimize the exchanged intra-domain traffic, while satisfying the delay constraint.

Simulations have been conducted by using the Neural Net Pattern Recognition Tool of MATLAB® for MLP solution. In addition, the solver provided by the integration of MATLAB® and IBM CPLEX Optimization Studio was used to derive the optimal DMEP solution.

6.2. Settings and metrics

As reference edge topology, we considered a metropolitan area network [62], [63] with 29 SDN edge nodes placed as shown in Fig. 1. Four upper-layer nodes, linked in a full meshed topology, are the roots of four binary trees, each one composed of three layers. Leaf nodes at the bottom behave as ingress nodes to which producers/clients are linked. The node in the middle of the mesh topology acts as the egress node that links the cloud with the edge domain. Nodes have different processing capabilities, as reported in Table 3, where the settings of the requested computing tasks are also indicated.

Four main classes of metrics are derived to comprehensively assess the performance of the considered supervised learning techniques.

Time complexity. It is aimed at estimating the efficiency and scalability of the considered supervised learning techniques.

ML-related metrics. After extracting the confusion matrices [64], starting from traditional binary classification metrics, a set of multi-class metrics are

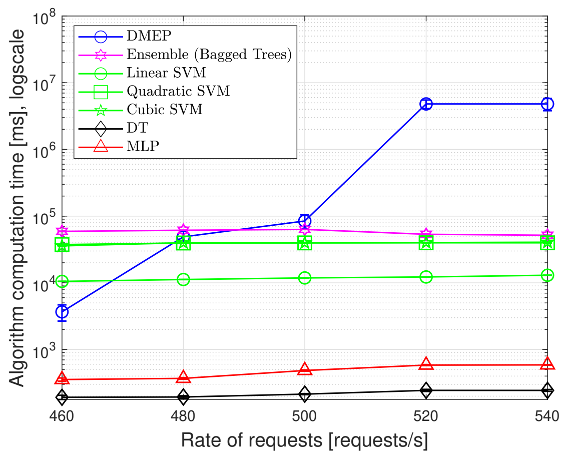


Figure 2: Comparison of the considered supervised learning approaches against the optimal solution (DMEP) in terms of computation time ($D_j^{max} \in [10, 100]$ ms
).

derived through common averaging techniques in the literature, including accuracy, macro average precision, macro average recall, and macro average F1 score[2] [65], [66].

Network usage metric. It is expressed in terms of *data exchanged* in the domain and computed as the overall amount of input data (in bytes) transmitted by producers to the selected executors multiplied by the number of hops traversed by such data in the network. This metric reflects the effectiveness of the proposal in minimizing the network usage.

Task-related metrics. They are specifically designed to assess how the requested tasks are treated by the softwarized intelligent edge and include:

- *Task offloading to the cloud*: it represents the percentage of requested computing tasks offloaded to the cloud.
- *Task deadline missing probability*: it represents the probability that the computation time of executed tasks exceeds their maximum computation delay.
- *Average queuing time*: it represents the average queuing time experienced by tasks before their execution at the selected edge node.
- *Average execution time*: it represents the average time needed for the execution of the requested tasks.

[2]It is worth noting that micro average precision, micro average recall and micro average F1 score are equal to the accuracy metric, thus they are not explicitly inserted in Table 4 for the sake of clarity.

- *Average computation time*: it is the sum of the average queuing time and the average execution time.

Among them, *task offloading to the cloud* and *task deadline missing probability* reflect the effectiveness of the compared approaches in offloading the tasks, while satisfying their deadline.

Values of the measured metrics are averaged over 200 runs and reported with 95% confidence intervals.

6.3. Time complexity analysis

The first set of results, reported in Fig. 2, aims to analyze the time complexity of the considered supervised learning techniques against the DMEP solution under high load conditions, versus the task requests rate. Fig. 2 gives also hints on the scalability of the algorithms, whose execution time unavoidably increases with the rate of requests. All simulation campaigns have been run on an Intel Xeon Gold 6240, 2.6 GHz-18 cores (CPU), 32 GB (RAM), 1 TB (HD).

As expected, the computation time of the DMEP solution dramatically increases with the rate of requests due to the enforced exhaustive search to find the optimal solution. It can be clearly observed that, among supervised learning techniques, DT and MLP are the most efficient solutions and well scale with the number of task requests, being both significantly below 1s. The other ML-based solutions provide too high algorithm computation time values which are all above 10s. Hence, their usage is not practically viable.

6.4. ML-related performance

In the following, DT and MLP algorithms are only considered given their lower time complexities w.r.t. the other supervised learning approaches, as the problem instance size increases.

Table 4 shows the ML-related performance metrics. It can be observed that the two compared techniques exhibit almost the same behaviour, with DT slightly outperforming MLP.

In particular, for both models, the accuracy, which measures the ratio of the number of correct task executor predictions to the total number of samples, is close to 0.6. Interestingly, all other metrics are above 0.6. Specifically, the macro average precision, which is computed as the arithmetic mean of all the precision values for the different classes (i.e., edge nodes), is 0.66 and 0.62 for DT and MLP, respectively. The macro average recall, derived as the arithmetic mean of all recall scores for different classes, is equal to 0.63 for both techniques. Macro average F1 score, measuring the harmonic mean of the macro average precision and recall, is higher than 0.63 for both models.

To better understand the performance results and their impact on task offloading, starting from the confusion matrices, we derived in Tables 5 and 6 the following probabilities for DT and MLP, respectively:

- P_c: is the probability that a given task is effectively placed by DT/MLP in the correct node (i.e., the one foreseen by the optimal solution).

- P_{next}: is the probability that a task, which should have been allocated in a given node according to the optimal solution, is actually placed by DT/MLP in an adjacent node in the same layer of the hierarchical topology.
- P_{up}: is the probability that a task, which should have been allocated in a given node according to the optimal solution, is actually placed by DT/MLP in an adjacent node in the immediately upper layer (excluding the cloud) of the hierarchical topology.
- P_{down}: is the probability that a task, which should have been allocated in a given node according to the optimal solution, is actually placed by DT/MLP in an adjacent node in the immediately lower layer of the hierarchical topology.
- P_{others}: is the probability that a task, which should have been allocated in a given node according to the optimal solution, is actually placed by DT/MLP in any other node except for those which are adjacent, in the immediately lower and upper layers.
- P_{cloud}: is the probability that a task, which should have been allocated in a given edge node according to the optimal solution, is actually placed by DT/MLP in the cloud.

Results are calculated for each level, from the cloud to the ingress nodes (level 4) of the considered hierarchical edge topology[3]. They clearly show that DT and MLP techniques sometimes may fail to select the same node as the optimal solution for the execution of a given task (the best one), as shown by the values of P_c columns in Tables 5 and 6. Notwithstanding, the executor selected by ML techniques is most likely adjacent to the best one (as captured by the values of P_{next}, P_{up}, P_{down}, columns in Tables 5 and 6) Seldom, the selected executor is more than one-hop far from the best one for each task request (as shown by the low values of P_{others} columns in Tables 5 and 6). Placing the task in an adjacent node does not significantly affect the metrics of interest, i.e., amount of exchanged data, computation time, by still well approximating the optimal solution (as it would be clearer in the following). This is because, in our scenario, adjacent nodes have the same computing capabilities and they are expected to provide similar computation times.

6.5. Supervised learning techniques Vs. the optimal solution

Fig. 3 reports the effectiveness metrics, when varying the rate of computing task requests for the following schemes: DMEP, DT, MLP. In addition, to better assess the impact of the delay constraint expressed in Eq. (7), curves labeled

[3]The probabilities P_{cloud}, P_{up}, and P_{down} are not defined for tasks which are expected to be placed into the cloud, into level 1 nodes and level 4 nodes, respectively.

Table 4: ML-related performance metrics.

Metrics	DT	MLP
Accuracy	0.59	0.58
Macro Average Precision	0.66	0.62
Macro Average Recall	0.63	0.63
Macro Average F1 Score	0.64	0.63

Table 5: Topology-aggregated confusion matrix for the DT ("-" stands for an ineligible probability).

	P_c	P_{next}	P_{up}	P_{down}	P_{others}	P_{cloud}
Cloud	0.924	0	0	0.049	0.027	-
Level 1	0.371	0	-	0.485	0.015	0.129
Level 2	0.570	0.236	0.058	0.113	0.022	0.005
Level 3	0.551	0.083	0.200	0.116	0.050	0
Level 4	0.691	0	0.194	-	0.114	0

as MEP are reported, which are representative of the formulated ILP problem without considering the maximum tolerable delay for task execution.

In particular, Fig. 3(a) shows that the amount of data exchanged into the edge domain reasonably increases as the rate of task requests increases, for all the compared schemes, because more input data need to traverse the network in order to reach the selected executors.

The proposed edge-based task placement solution, both in its optimal formulation, DMEP, as well as when solved through DT and MLP, achieves a significantly lower amount of exchanged data compared to a baseline scheme, the *CloudP* approach, which foresees to transfer input data for each task to the remote cloud, where tasks are all executed.

Interestingly, both DT and MLP satisfactorily well approximate the performance of the optimal DMEP solution.

To meet computing delay constraints of tasks which cannot be allocated to the edge, all the schemes, except for MEP, offload a percentage of tasks to the cloud (up to a maximum of about 3% for MLP), as shown in Fig. 3(b). This is because, if not offloaded remotely, tasks could be queued for a too long time before being executed by edge nodes which exhibit significantly lower computing capabilities compared to the cloud.

Contrarily to what happens with DMEP, where the delay bounds are assured by hard constraints in the mathematical formulation of the optimization model (Eq. (7)), when it comes to MLP and DT meeting such delay constraints depends solely on the ability of the pursued approach to infer the correct placement for each task. Therefore, although both DT and MLP achieve a quite satisfactory precision, they may fail to predict the optimal placement for each task also incurring in occasional violation of the delay bounds (see Fig. 3(c)). Nonetheless such misclassification errors, the task deadline missing probability values are in the order of 0.1, in the worst case, and well below the values

Table 6: Topology-aggregated confusion matrix for the MLP ("-" stands for an ineligible probability).

	P_c	P_{next}	P_{up}	P_{down}	P_{others}	P_{cloud}
Cloud	0.962	0	0	0.027	0.011	-
Level 1	0.347	0	-	0.434	0.030	0.189
Level 2	0.550	0.186	0.068	0.150	0.045	0.007
Level 3	0.539	0.089	0.164	0.132	0.076	0
Level 4	0.695	0	0.163	-	0.141	0

achieved by MEP, which does not consider delay constraints in its formulation. In doing so, MEP prioritizes task execution at the edge - no tasks are offloaded to the cloud (Fig. 3(b)) - and, hence, it achieves the lowest amount of exchanged data (Fig. 3(a)).

There are no remarkable differences between DMEP, MLP and DT in terms of queueing delay (Fig. 4(a)), execution delay (Fig. 4(b)) and computation delay (Fig. 4(c)). Moreover, the metrics are not significantly affected by the task request rate.

Contrarily, in MEP, there is a faster increase in the average queuing delay compared to the other solutions, when the rate of requests increases (Fig. 4(a)). The average execution delay, instead, decreases when varying the rate of task requests (Fig. 4(b)). This trend is due to the fact that MEP is agnostic of delay constraints, thus it loads the nodes from the less capable to the more powerful ones, and task requests may be queued for a long time before being executed by edge nodes. Such trends are jointly captured by the computation delay shown in Fig. 4(c).

Curves for the CloudP approach are also reported for the execution delay and computation delay metrics (Fig. 4(b) and Fig. 4(c), respectively) with values which approach 0, given the virtually unlimited computing capability of the cloud and no queuing delay experienced before task execution.

Furthermore, results for all the measured metrics report trends without any statistically meaningful variation with the increasing number of task requests, as shown by the small confidence intervals.

6.6. Impact of delay constraints

Misclassification issues of DT and MLP can get critical when tasks exhibit strict delay constraints. If tasks are not executed within their deadline, their output may be meaningless for the requesting users and their execution may uselessly waste resources. To fix them, we improved the conceived algorithms through a post-processing procedure to be performed by the Task placement policy deployed at the SDN controller. In front of a temporary allocation provided by the implemented ML-based strategy for a given task request, the network application in charge of task placement checks if the delay constraints are met for it. If it is not the case, the SDN controller decides to directly place that task into the cloud in order to satisfy the delay constraint, although at the cost of a larger data exchange into the edge domain. We refer to the resulting enhanced schemes as *Constrained-MLP* and *Constrained-DT* for the MLP and DT strategies, respectively.

The last set of results compares the enhanced schemes against the basic approaches under different delay settings. In particular, without loss of generality, values of the maximum delay constraints are uniformly distributed in two different ranges $[10, 50]$ ms and $[50, 150]$ ms, with bound values aligned with those commonly considered in the literature, [67], [68], [69], to cover a wide set of vertical application scenarios [70], with different requirements, spanning from factory automation to automotive.

Figure 5(a) shows that the enhanced schemes (solid lines) exhibit a larger amount of exchanged data compared to the basic ones (dashed lines). This is because they offload a higher percentage of tasks to the cloud (about 35% for MLP and DT with 500 requests/s as shown in Fig. 5(b)) to better satisfy delay constraints. Figure 5(c) shows that the task deadline missing probability is close to 0 whereas it approaches 0.45 and 0.4 with 500 requests/s, respectively for the basic MLP and DT strategies.

Fig. 6 reports delay values for the compared schemes. It can be observed that the benefits in meeting the delay constraints of the conceived enhanced schemes get more remarkable as these constraints get stricter and the rate of task requests increases (Fig. 6(a) and Fig. 6(c)). Indeed, higher differences can be observed between the basic and the constrained approaches. This is because as delay constraints get stricter, the basic approaches may experience more misclassification issues.

7. Conclusions and future works

Allocating computing tasks with strict performance demands to heterogeneous edge nodes with limited capabilities is very challenging. Since task allocation is an online problem, it cannot be efficiently solved by standard optimization solvers. In this work, we implemented a set of supervised learning techniques, typically leveraged in the literature to address multi-class classification problems and, more specifically, task offloading at the edge. We customized these techniques to solve the formulated task placement optimization problem.

Although such techniques require heavy training procedures, the latter ones can be performed offline and hence, they do not affect the task placement decision time. In particular, among tested techniques, MLP and DT prove to be the most efficient solutions in terms of execution time for the task allocation decision, with DT achieving slightly lower execution time values than MLP

Results collected under a wide variety of settings showcase that, besides being computationally-efficient, MLP and DT provide a good approximation of the solution achieved through a standard optimization solver. Indeed, they are both satisfactorily successful in minimizing the amount of data exchanged in the edge domain which is the objective of the formulated optimization problem. MLP is slightly better performing in satisfying the task deadline compared to DT.

Furthermore, enhanced versions of the aforementioned techniques have been devised to face possible misclassification issues due to the inability to meet the task delay constraints, which get particularly critical for tasks with stricter demands. Whenever the task placement network application at the SDN controller realizes that the placement decision from the ML technique does not meet the task delay constraint, regardless of the suggested executor, the task is offloaded to the cloud. With the conceived enhancements, the two techniques perform almost equally.

The work has room for further improvements. Under highly dynamic edge environments, task offloading decisions may face unseen scenarios, e.g., in terms of tasks requests rate, for which relevant labeled data may be difficult to obtain. Semi-supervised learning (SSL) techniques can help solve this issue, thus complementing our solution. Among the many SSL techniques, extensively surveyed for instance in [71], wrapper methods can be coupled with the ML techniques proven to be the best performing in our study, i.e., DT and MLP. Of course, the applicability in our context of such SSL techniques, which can be used with virtually any supervised base learner, requires further investigation to understand the performance improvements and the price to pay to achieve them. For instance, self-training, the most basic of pseudo-labelling approaches, would take a longer time, since it requires additional iterations to obtain predictions for the unlabelled data points in addition to the training of the model on labelled data.

References

[1] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A survey on computation offloading modeling for edge computing, Journal of Network and Computer Applications 169 (2020) 102781.

[2] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, M. Zorzi, Toward 6G networks: Use cases and technologies, IEEE Communications Magazine 58 (3) (2020) 55–61.

[3] A. Al-Shuwaili, O. Simeone, Energy-efficient resource allocation for mobile edge computing-based augmented reality applications, IEEE Wireless Communications Letters 6 (3) (2017) 398–401.

[4] Q. Fan, N. Ansari, Application aware workload allocation for edge computing-based IoT, IEEE Internet of Things Journal 5 (3) (2018) 2146–2153.

[5] D. M. Manias, M. Jammal, H. Hawilo, A. Shami, P. Heidari, A. Larabi, R. Brunner, Machine learning for performance-aware virtual network function placement, in: IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6.

[6] H. Guo, J. Liu, J. Lv, Toward intelligent task offloading at the edge, IEEE Network 34 (2) (2019) 128–134.

[7] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, X. Chen, Convergence of edge computing and deep learning: A comprehensive survey, IEEE Communications Surveys & Tutorials 22 (2) (2020) 869–904.

[8] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective, Computer Networks 182 (2020) 107496.

[9] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu, A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges, IEEE Communications Surveys & Tutorials 21 (1) (2018) 393–430.

[10] S. Yu, X. Wang, R. Langar, Computation offloading for mobile edge computing: A deep learning approach, in: IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), 2017, pp. 1–6.

[11] I. Kunze, K. Wehrle, D. Trossen, M.-J. Montpetit, Use cases for in-network computing, IETF, Internet-Draft (2021).

[12] M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, A. Iera, SDN-managed provisioning of named computing services in edge infrastructures, IEEE Transactions on Network and Service Management 16 (4) (2019) 1464–1478.

[13] G. Lia, M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, Optimal placement of delay-constrained in-network computing tasks at the edge with minimum data exchange, in: IEEE 4th 5G World Forum (5GWF), 2021, pp. 481–486.

[14] D. Kreutz, et al., Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[15] D. Rahbari, M. Nickray, Task offloading in mobile fog computing by classification and regression tree, Peer-to-Peer Networking and Applications 13 (1) (2020) 104–122.

[16] C. Sonmez, C. Tunca, A. Ozgovde, C. Ersoy, Machine learning-based workload orchestrator for vehicular edge computing, IEEE Transactions on Intelligent Transportation Systems 22 (4) (2020) 2239–2251.

[17] Y. Mao, C. You, J. Zhang, K. Huang, K. B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE communications surveys & tutorials 19 (4) (2017) 2322–2358.

[18] Q.-V. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W.-J. Hwang, Z. Ding, A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art, IEEE Access 8 (2020) 116974–117017.

[19] J. Plachy, Z. Becvar, E. C. Strinati, N. di Pietro, Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users, IEEE Transactions on Network and Service Management 18 (2) (2021) 2089–2106.

[20] A. Rago, G. Piro, G. Boggia, P. Dini, Anticipatory allocation of communication and computational resources at the edge using spatio-temporal dynamics of mobile users, IEEE Transactions on Network and Service Management 18 (4) (2021) 4548–4562.

[21] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, Y. Wang, Cloudlet placement and task allocation in mobile edge computing, IEEE Internet of Things Journal 6 (3) (2019) 5853–5863.

[22] B. Nour, S. Mastorakis, A. Mtibaa, Compute-less networking: Perspectives, challenges, and opportunities, IEEE Network 34 (6) (2020) 259–265.

[23] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, N. Kato, Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective, IEEE Communications Surveys & Tutorials 22 (1) (2019) 38–67.

[24] S. M. Araújo, F. S. de Souza, G. R. Mateus, A hybrid optimization-Machine Learning approach for the VNF placement and chaining problem, Computer Networks 199 (2021) 108474.

[25] J. Guo, Z. Song, Y. Cui, Z. Liu, Y. Ji, Energy-efficient resource allocation for multi-user mobile edge computing, in: IEEE Global Communications Conference (GLOBECOM), 2017, pp. 1–7.

[26] S. Wu, W. Xia, W. Cui, Q. Chao, Z. Lan, F. Yan, L. Shen, An efficient offloading algorithm based on support vector machine for mobile edge computing in vehicular networks, in: IEEE International Conference on Wireless Communications and Signal Processing (WCSP), 2018, pp. 1–6.

[27] X. Li, Z. Xu, F. Fang, Q. Fan, X. Wang, V. C. Leung, Task offloading for deep learning empowered automatic speech analysis in mobile edge-cloud computing networks, IEEE Transactions on Cloud Computing (2022).

[28] Y. Wang, L. Gao, J. Ren, R. Cao, H. Wang, J. Zheng, Q. Gao, Ato-edge: Adaptive task offloading for deep learning in resource-constrained edge computing systems, in: IEEE ICPADS, 2021, pp. 153–160.

[29] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D. I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Communications Surveys & Tutorials 21 (4) (2019) 3133–3174.

[30] L. Weissbart, Performance analysis of multilayer perceptron in profiling side-channel analysis, in: International Conference on applied cryptography and network security, Springer, 2020, pp. 198–216.

[31] A. C. Baktir, A. Ozgovde, C. Ersoy, How can edge computing benefit from software-defined networking: A survey, use cases, and future directions, IEEE Communications Surveys & Tutorials 19 (4) (2017) 2359–2391.

[32] A. Wang, Z. Zha, Y. Guo, S. Chen, Software-defined networking enhanced edge computing: A network-centric survey, Proceedings of the IEEE 107 (8) (2019) 1500–1519.

[33] S.-C. Lin, K.-C. Chen, A. Karimoddini, Sdvec: Software-defined vehicular edge computing with ultra-low latency, IEEE Communications Magazine 59 (12) (2021) 66–72.

[34] J. Zhang, H. Guo, J. Liu, Adaptive task offloading in vehicular edge computing networks: a reinforcement learning based scheme, Mobile Networks and Applications 25 (5) (2020) 1736–1745.

[35] N. Kiran, C. Pan, Y. Changchuan, Reinforcement learning for task offloading in mobile edge computing for sdn based wireless networks, in: 2020 Seventh International Conference on Software Defined Systems (SDS), IEEE, 2020, pp. 268–273.

[36] P. Zhou, G. Wu, B. Alzahrani, A. Barnawi, A. Alhindi, M. Chen, Reinforcement learning for task placement in collaborative cloud-edge computing, in: IEEE Global Communications Conference (GLOBECOM), 2021, pp. 1–6.

[37] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, R. P. Liu, Energy-efficient admission of delay-sensitive tasks for mobile edge computing, IEEE Transactions on Communications 66 (6) (2018) 2603–2616.

[38] S.-W. Ko, K. Han, K. Huang, Wireless networks for mobile edge computing: Spatial modeling and latency analysis, IEEE Trans. on Wireless Comm. 17 (8) (2018) 5225–5240.

[39] T. Zhao, S. Zhou, X. Guo, Z. Niu, Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing, in: IEEE ICC 2017, pp. 1–7.

[40] C. Jiang, Y. Chen, Q. Wang, K. R. Liu, Data-driven stochastic scheduling and dynamic auction in IaaS, in: IEEE GLOBECOM, pp. 1–6.

[41] M. Zukerman, Introduction to queueing theory and stochastic teletraffic models, arXiv preprint arXiv:1307.2968 (2013).

[42] D. G. Cattrysse, L. N. Van Wassenhove, A survey of algorithms for the generalized assignment problem, European journal of operational research 60 (3) (1992) 260–272.

[43] P. C. Sen, M. Hajra, M. Ghosh, Supervised classification algorithms in machine learning: A survey and review, in: Emerging technology in modelling and graphics, Springer, 2020, pp. 99–111.

[44] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and regression trees, Routledge, 2017.

[45] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, Advances in neural information processing systems 25 (2012).

[46] L. Breiman, Bagging predictors, Machine learning 24 (2) (1996) 123–140.

[47] M. F. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Neural networks 6 (4) (1993) 525–533.

[48] P. Y. Taser, Application of bagging and boosting approaches using decision tree-based algorithms in diabetes risk prediction, in: Multidisciplinary Digital Publishing Institute Proceedings, Vol. 74, 2021, p. 6.

[49] S. Haykin, Neural networks: a comprehensive foundation, The Knowledge Engineering Review 13 (4) (1999) 409–412.

[50] E. Mayoraz, E. Alpaydin, Support vector machines for multi-class classification, in: International Work-Conference on Artificial Neural Networks, Springer, 1999, pp. 833–842.

[51] G. Aurélien, Hands-on machine learning with scikit-learn & tensorflow, Geron Aurelien (2017).

[52] H. Huang, H. V. Burton, Classification of in-plane failure modes for reinforced concrete frames with infills using machine learning, Journal of Building Engineering 25 (2019) 100767.

[53] M. Adhikari, M. Mukherjee, S. N. Srirama, DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing, IEEE Internet of Things Journal 7 (7) (2019) 5773–5782.

[54] X. Hao, R. Zhao, T. Yang, Y. Hu, B. Hu, Y. Qiu, A risk-sensitive task offloading strategy for edge computing in industrial internet of things, EURASIP Journal on Wireless Communications and Networking 2021 (1) (2021) 1–18.

[55] J. Yang, B. Ai, I. You, M. Imran, L. Wang, K. Guan, D. He, Z. Zhong, W. Keusgen, Ultra-reliable communications for industrial internet of things: Design considerations and channel modeling, IEEE Network 33 (4) (2019) 104–111.

[56] M. Masoudi, C. Cavdar, Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption, IEEE Transactions on Mobile Computing 20 (12) (2020) 3324–3337.

[57] L. Leng, J. Li, H. Shi, Y. Zhu, Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing, Multimedia Tools and Applications 80 (19) (2021) 29163–29175.

[58] L. Huang, X. Feng, C. Zhang, L. Qian, Y. Wu, Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing, Digital Communications and Networks 5 (1) (2019) 10–17.

[59] J. Almutairi, M. Aldossary, H. A. Alharbi, B. A. Yosuf, J. M. Elmirghani, Delay-optimal task offloading for uav-enabled edge-cloud computing systems, IEEE Access (2022).

[60] Z. Zhang, N. Wang, H. Wu, C. Tang, R. Li, Mr-dro: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments, IEEE Internet of Things Journal (2021).

[61] A. Ali-Eldin, B. Wang, P. Shenoy, The hidden cost of the edge: a performance comparison of edge and cloud latencies, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021, pp. 1–12.

[62] X. Xu, Y. Li, T. Huang, Y. Xue, K. Peng, L. Qi, W. Dou, An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks, Journal of Network and Computer Applications 133 (2019) 75–85.

[63] X. Chen, W. Liu, J. Chen, J. Zhou, An edge server placement algorithm in edge computing environment, in: IEEE ICAIT, 2020, pp. 85–89.

[64] M. Naser, A. H. Alavi, Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences, Architecture, Structures and Construction (2021) 1–19.

[65] I. Markoulidakis, I. Rallis, I. Georgoulas, G. Kopsiaftis, A. Doulamis, N. Doulamis, Multiclass confusion matrix reduction method and its application on net promoter score classification problem, Technologies 9 (4) (2021) 81.

[66] M. Grandini, E. Bagli, G. Visani, Metrics for multi-class classification: an overview, arXiv preprint arXiv:2008.05756 (2020).

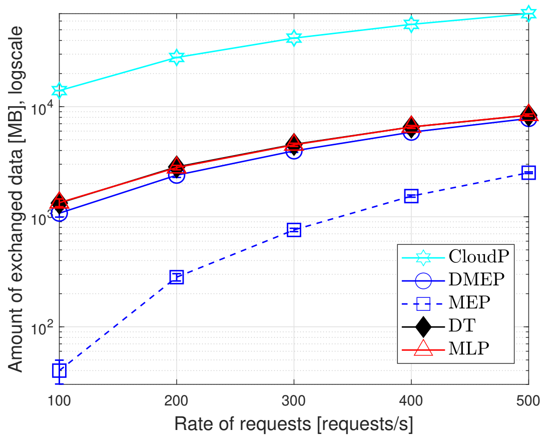
[67] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, X. Shen, Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach, IEEE Transactions on Mobile Computing 20 (3) (2019) 939–951.

[68] K. Toczé, S. Nadjm-Tehrani, Orch: Distributed orchestration framework using mobile edge devices, in: IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), 2019, pp. 1–10.

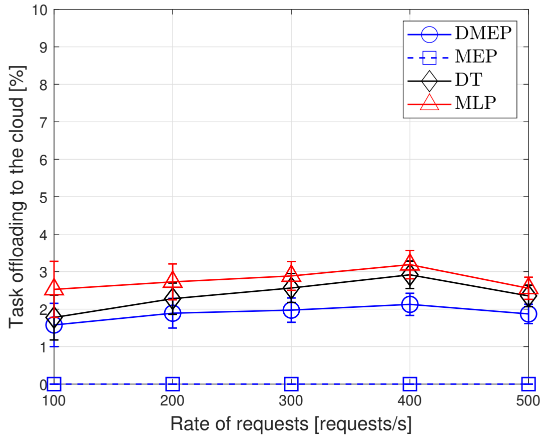
[69] Y. Sun, S. Zhou, J. Xu, Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks, IEEE Journal on Selected Areas in Communications 35 (11) (2017) 2637–2646.

[70] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, C. Assi, Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing, IEEE Journal on Selected Areas in Communications 37 (3) (2019) 668–682.

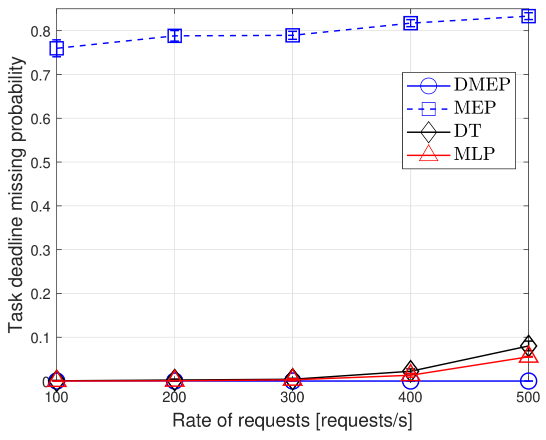
[71] J. E. Van Engelen, H. H. Hoos, A survey on semi-supervised learning, Machine Learning 109 (2) (2020) 373–440.



(a) Exchanged data.

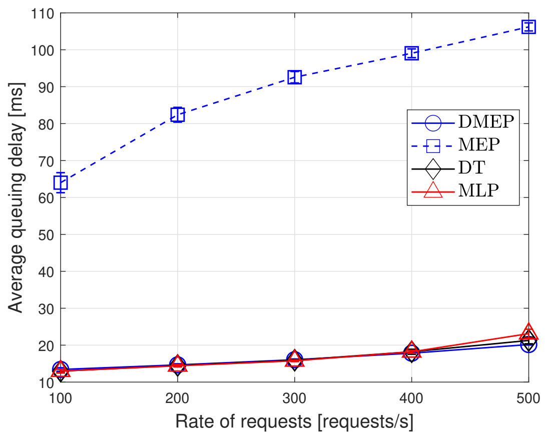


(b) Task offloading to the cloud.

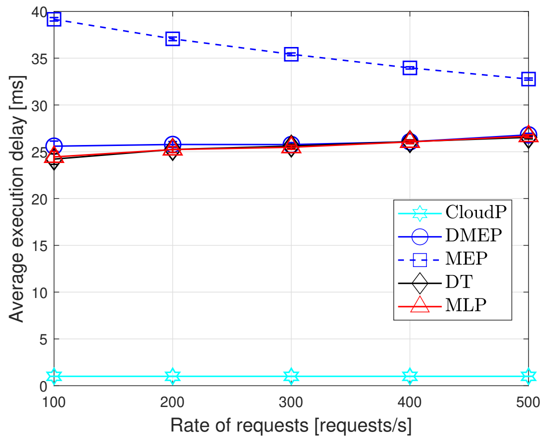


(c) Task deadline missing probability.

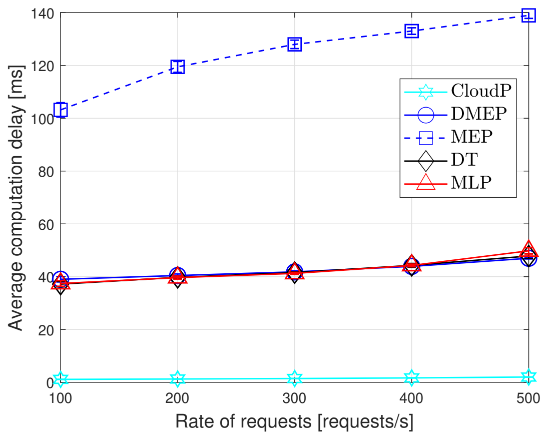
Figure 3: Effectiveness metrics ($D_j^{max} \in [10, 100]$ ms).



(a) Average queuing time.

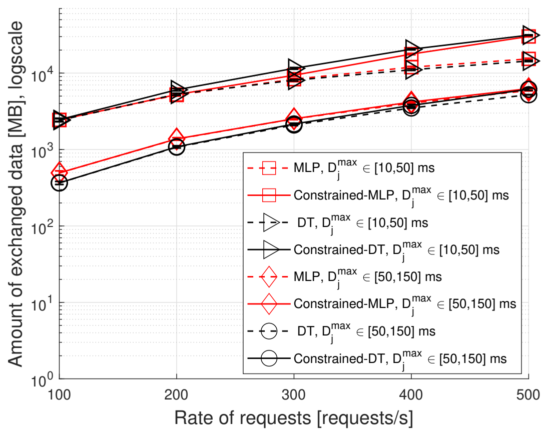


(b) Average execution time.

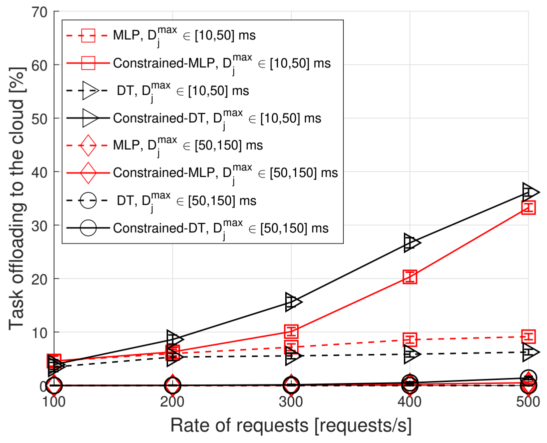


(c) Average computation time.

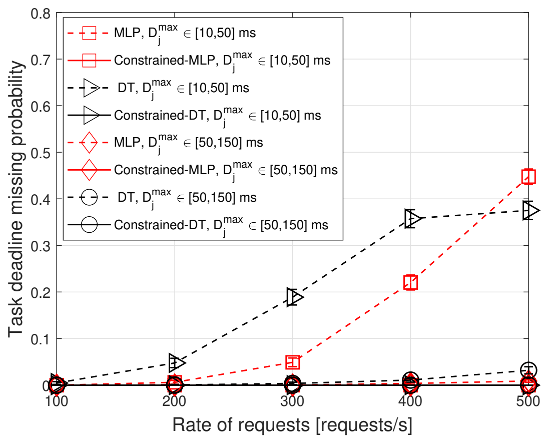
Figure 4: Delay metrics ($D_j^{max} \in [10, 100]$ ms
).



(a) Exchanged data.

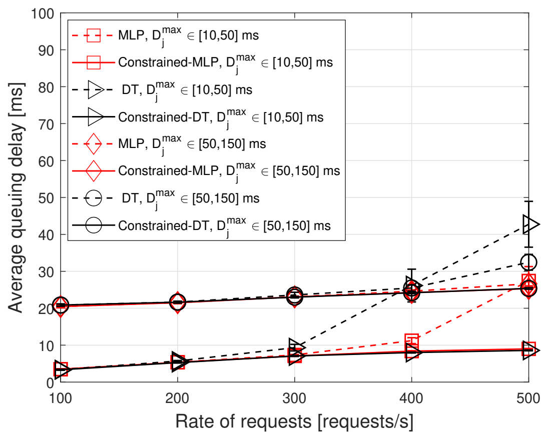


(b) Task offloading to the cloud

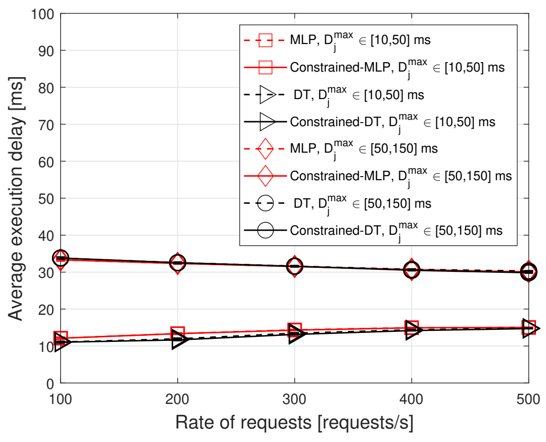


(c) Task deadline missing probability.

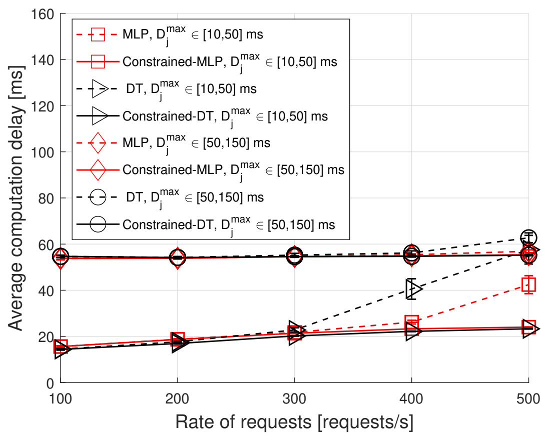
Figure 5: MLP Vs. DT: effectiveness metrics.



(a) Average queuing time.



(b) Average execution time.



(c) Average computation time.

Figure 6: MLP Vs. DT: delay metrics.