DOCTORAL SCHOOL
MEDITERRANEA UNIVERSITY OF REGGIO CALABRIA

DEPARTMENT OF INFORMATION ENGINEERING, INFRASTRUCTURES
AND SUSTAINABLE ENERGY
(DIIES)

PHD IN
INFORMATION ENGINEERING

S.S.D. ING-INF/03
XXXV CYCLE

TASK ALLOCATION IN A SOFTWARIZED EDGE NETWORK

CANDIDATE
Gianmarco LIA

ADVISOR
Prof. Giuseppe RUGGERI

COORDINATOR
Prof. Antonio IERA

REGGIO CALABRIA, GENNUARY 2023

GIANMARCO LIA

TASK ALLOCATION IN A SOFTWARIZED EDGE NETWORK

The Teaching Staff of the PhD course in

INFORMATION ENGINEERING

consists of:

Antonio IERA (coordinator)
Pier Luigi ANTONUCCI
Giuseppe ARANITI
Francesco BUCCAFURRI
Claudia CAMPOLO
Giuseppe COPPOLA
Mariantonia COTRONEI
Lorenzo CROCCO
Dominique DALLET
Claudio DE CAPUA
Francesco DELLA CORTE
Giuliana FAGGIO
Gioia FAILLA
Fabio FILIANOTI
Patrizia FRONTERA
Sofia GIUFFRÈ
Giorgio GRADITI
Voicu GROZA
Tommaso ISERNIA
Gianluca LAX
Aimè LAY EKUAKILLE
Gaetano LICITRA
Antonella MOLINARO
Andrea MORABITO
Carlo Francesco MORABITO
Giacomo MORABITO
Rosario MORELLO
Fortunato PEZZIMENTI
Filippo PRATICÒ
Domenico ROSACI
Giuseppe RUGGERI
Mariateresa RUSSO
Antonino VITETTA


UNIVERSITÀ DEGLI STUDI DI REGGIO CALABRIA
Mediterranea

To my dad

Acknowledgments

This research was partially funded by Regione Calabria, POR Calabria FESR/FSE 2014-2020 Linea A (Asse 12 Azione 10.5.6).

Abstract

Applications like augmented/extended reality (AR, XR), autonomous driving, control and automation procedures in Industry 4.0 could have strict connectivity requirements (e.g., latency, throughput) that current networks can not guarantee. For their successful implementation, such applications may also require additional computational resources (e.g., for object detection, data aggregation, inference decisions). However, the resources they require may not be available in the end devices (e.g., smartphones, sensors, actuators). Multi-access Edge Computing (MEC), among the enabling technologies for *fifth generation (5G) and beyond systems*, aims to support these requirements by moving "cloud services" to network nodes closer to the end-devices. Thus, it allows to decrease latency and the wastage of the network bandwidth required to transfer input data to be processed.

In this context, it is necessary to define self-aware solutions able to decide how and where to allocate the services, in order to satisfy the performance requirements, taking into account the heterogeneity and dynamics of computational, storage, and network resources, distributed on the edge nodes.

Moreover, Edge Intelligence (EI), i.e., the edge orchestrated by Artificial Intelligence (AI) techniques (e.g., machine learning, deep neural networks, etc.), is already regarded as one of the main missing pieces in 5G networks in order to support the performance, new unprecedented functionalities, and new challenging and demanding services of future sixth generation (6G) networks.

This thesis contributes to this research area with the design of a novel centralized framework based on the synergy of innovative networking technologies, for a joint and judicious orchestration of computational and network resources at the edge.

The contributions of the study encompass: i) the design of a computing task allocation strategy which has the aim to minimize the amount of data exchanged within the edge domain while ensuring tasks' QoS requirements and its mathematical formulation as ILP optimization problem, called DMEP, ii) its resolution through a set of ML algorithms to judiciously orchestrate the placement of delay-constrained computing tasks in the softwarized edge domain, and iii) the design and formulation of a new computing task allocation strategy that takes into account the "compute reuse" of previously computed tasks's output. A heuristic is also implemented to solve the formulated problems. Each proposal is validated and evaluated against different benchmark solutions under a variety of simulation settings.

Sommario

Applicazioni come la realtà aumentata/estesa (AR, XR), la guida autonoma, procedure di controllo e automazione nell'Industria 4.0, hanno requisiti di connettività stringenti (ad esempio, latenza, throughput) che le reti attuali non possono garantire. Per la loro implementazione, tali applicazioni richiedono anche risorse computazionali (ad esempio, per il rilevamento degli oggetti, l'aggregazione dei dati e le decisioni di inferenza). Le risorse richieste potrebbero non essere disponibili nei dispositivi finali (ad esempio, smartphone, sensori, attuatori).

Il Multi-access Edge Computing (MEC), tra le tecnologie abilitanti per i sistemi di quinta generazione (5G), mira a supportare questi requisiti spostando i servizi cloud sui nodi di rete più vicini ai dispositivi finali, in modo da ridurre la latenza e lo spreco di banda di rete necessari per trasferire i dati da elaborare. In questo contesto è necessario definire soluzioni in grado di decidere come e dove allocare i servizi, al fine di soddisfare i requisiti e tenendo conto dell'eterogeneità e della dinamicità delle risorse computazionali, di memorizzazione e di rete, distribuite sui nodi edge.

Inoltre, l'Edge Intelligence (EI), ossia l'edge orchestrato grazie all'ausilio di tecniche di Intelligenza Artificiale (AI) (ad esempio, machine learning, deep neural networks, ecc.), è già considerato uno dei principali tasselli mancanti nelle reti 5G e di supportare le prestazioni, le nuove funzionalità e i nuovi servizi delle future reti 6G.

Questa tesi contribuisce in quest'area di ricerca con la progettazione di un framework centralizzato basato sulla sinergia di tecnologie di rete innovative, per un'orchestrazione congiunta di risorse computazionali e di rete in un contesto edge.

Lo studio comprende la progettazione di strategie di allocazione di task computazionali all'edge, la formulazione matematica di tali strategie come algoritmi di ottimizzazione, la loro risoluzione attraverso euristiche e algoritmi basati sul machine learning, la loro validazione rispetto a soluzioni di benchmark in diversi scenari di simulazione.

I contributi del lavoro di tesi comprendono: i) la progettazione di una strategia di allocazione di task computazionali all'edge che ha l'obiettivo di minimizzare la quantità di dati scambiati all'interno del dominio edge, garantendo al contempo i requisiti di QoS, e la sua formulazione matematica come problema di ottimizzazione ILP, chiamato DMEP, ii) la sua risoluzione attraverso una serie di algoritmi ML per orchestrare il posizionamento dei task computazionali con vincoli di ritardo nel dominio edge softwarizzato, e iii) la progettazione e la formulazione di una nuova strategia di allocazione dei task computazionali che tiene conto del "compute reuse" dei risultati dei compiti precedentemente processati. Viene inoltre implementata un'euristica per risolvere i problemi formulati. Ciascuna proposta è convalidata e valutata rispetto a diverse soluzioni di riferimento con diverse impostazioni di simulazione.

Contents

List of Figures

List of Tables

Abbreviations

AP	Access point
BS	Base Station
ICN	Information Centric Networking
ISG	Industry Specification Group
IRTF	Internet Research Task Force
IETF	Internet Engineering Task Force
CNN	Convolutional Neural Network
COIN	Computing in the Network
ETSI	European Telecommunications Standards Institute
5G	fifth generation
6G	sixth generation
AI	Artificial Intelligence
AR	Augmented Reality
ASA	Automatic Speech Analysis
BS	Base Station
BTs	Bagged Trees
CART	Classification and Regression Tree
DL	Deep Learning
DMEP	Delay-constrained Minimum data Edge task Placement
DNN	Deep Neural Network
DT	Decision Tree
FL	Federated Learning
GAP	Generalized Assignment Problem
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastracture as a service
ICT	Information and Communications Technology
ILP	Integer Linear Programming
IMT	International Mobile Telecommunications
IoT	Internet of Things
IT	Information Technology

ITU-T	International Telecommunication Union Telecommunication
LTE	Long Term Evolution
MCC	Mobile Cloud Computing
MEC	Multi-acces Edge Computing
MLP	Multi-Layer Perceptron
MINLP	Mixed Integer Non Linear Programming
ML	machine learning
MNO	Mobile Network Operator
OF	OpenFlow
PaaS	Platform as a service
PoP	Points of Presence
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
SaaS	Software as a service
SDN	Software-defined Networking
SFC	Service Function Chain
SVM	Support Vector Machine
UE	User Equipment
UHD	Ultra-high definition
VNF-PC	Virtual Network Function Placement and Chainingservice function chain
VNF	Virtual Network Function

1

Introduction

The development of new paradigms and modern technologies, that are able to deliver innovative services to users (autonomous driving, augmented reality, *smart envirorment*), has taken a high priority in research and development (R&D) activities in the scientific community, companies, and European and international standardization bodies. Such services have highly stringent network requirements that conventional networks can not support. Fifth-generation (5G) systems have set out to support such services.

The *International Telecommunication Union* (ITU) has classified three classes of service that such systems focus on:

1. *massive Machine Type Communications* (mMTC);
2. *enhanced Mobile BroadBand* (eMBB);
3. *Ultra Reliable and Low Latency Communications* (URLLC).

Specifically, mMTC refers to services that involve a huge number of devices connecting to the network without the need for human interaction, e.g., sensors, actuators and other *Internet of Things* (IoT) devices; eMBB refers to services that require high bandwidth and high throughput, for example augmented reality services or distribution of high-definition (4K/8K) content; and URLLC includes services with very high reliability and very low latency requirements. Such services can be autonomous driving or remote surgery. It is worth noting the term International Mobile Telecommunications (IMT) is the generic term used by the ITU community to designate broadband mobile systems.

In 2015 IMT-2020, thus 5G, was expected to provide far more enhanced capabilities than those described before for IMT-Advanced, and these enhanced capabilities could have been regarded as new capabilities of future IMT. Moreover, a broad variety of requirements, tightly coupled with intended usage scenarios and applications for IMT-2020 was envisioned. Hence, different service classes result in a great diversity/variety of requirements. The key design principles were flexibility and diversity to serve many different use cases and scenarios, for which the capabilities of IMT-2020, depicted in Fig. 1.1, have different relevance and applicability.

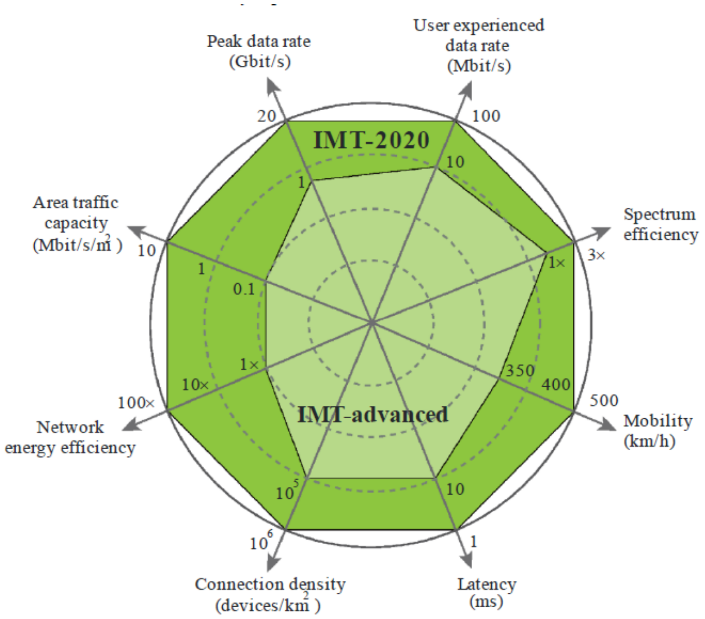


Fig. 1.1. 5G service requirements [1]

In addition to these capabilities, today's and near-future applications require storage and processing resources that are typically not present within the mobile devices that request the service.

Although much progress has been made in technologies and industrialization processes, smartphones, laptops, IoT devices, sensors, and actuators still have limited storage, computation, and battery capabilities that do not allow the fruition of innovative compute-intensive applications.

Different technologies have been proposed in the literature to overcome the problems of limited device resources and therefore, to enable the provisioning of the aforementioned services. Over the years, there has been a shift from an approach based on Cloud Computing technologies that provide users with computational resources on remote data centers to the "edge computing" approach, in which these resources are allocated in a distributed and heterogeneous manner on the network nodes closest to the end users (i.e., edge nodes), thus decreasing the latency along the consumer-supplier path of the services and the amount of packets exchanged in the core network.

Several issues remain open in such an approach, first of all how to determine the allocation of resources on the various nodes, secondly in which nodes to allocate services, in other words where to store contents, to meet any future demands, and where to process them.

Although the Fifth Generation (5G) is still being deployed globally, it is already time for academia and business to focus on the Sixth Generation (6G) systems in order to meet the future demands for Information and Communications Technology (ICT) in 2030 [10].

In the International Telecommunication Union Telecommunication (ITU-T) standardization sector, a focus group dubbed Technologies for Network 2030 was established in July

2018. The group wants to research how networks will function in 2030 and beyond [11], when it is anticipated that they will be able to handle cutting-edge future scenarios including holographic communications, pervasive intelligence, tactile internet, multi-sense experiences, and digital twins.

In its recent Horizon 2020 calls, ICT-20 5G Long Term Evolution and ICT-52 Smart Connectivity Beyond 5G, the European Commission took the initiative to support research activities beyond 5G. As a result, a number of innovative research projects for important 6G technologies were launched at the beginning of 2020.

Within the context of pervasive intelligence, the Edge Intelligence (EI), which exploits the interplay of edge computing and artificial intelligence, is becoming a key concept for the development of the beyond-5G systems. On one hand, Edge for AI, aiming to facilitate the deployment of AI services (think about Siri of Apple, or Tesla car) by edge computing, has received significant attention. On the other hand, Deep Learning (DL), a technique of artificial intelligence, can be used into edge computing frameworks for dynamic, adaptive edge orchestration, maintenance and management.

In this thesis the task offloading problem has been faced. In particular, a new strategy for placing *CPU-intensive* and *delay-constrained* reusable computing tasks into a softwarized edge domain made of several network nodes augumented with processing capabilities has been designed. Within the thesis, the proposed strategy has been formulated as an Integer Linear Programming (ILP) problem, called Delay-constrained Minimum data Edge task Placement (DMEP) aimed at ensuring that tasks' Quality of Service (QoS) requirements are met, but also at minimizing the network resources usage by reducing the amount of data exchanged within the edge domain.

The formulated problem was initially solved using optimal standard solvers. Despite the fact that the found solution is the one that minimizes the objective function in the best possible way, the proven NP-hard nature of the problem makes scaling and practicality difficult.

Hence, different techniques have so been implemented to address these problems producing near-optimal results, which contribute to advance the state of the art in the edge computing domain.

This thesis work will be structured as follows:

- **Chapter 2**: the chapter presents the main features of the Cloud Computing paradigm and its evolution toward "Mobile Cloud Computing" and "ulti-access Edge Computing" (MEC). The latter moves the execution of services as close to the user as possible, producing several benefits for both the network and the user. These features make the MEC paradigm a key technology for 5G and 6G systems, ensuring low latency in accessing computation and storage resources and relaxing the core network. Several MEC use cases are illustrated. Furthermore, the recent concept of Edge Intelligence is explained to motivate the works of the next chapters.

- **Chapter 3**: in this chapter, the task allocation challenge at the edge is discussed. Firstly, focus will be on the main context and the state of the art on this topic. Then, the first contribution of the thesis is presented, by formulating an optimization problem as an ILP problem (called DMEP) with the objective to minimize the amount of data exchanged within the edge domain while ensuring tasks' QoS requirements (formulated as maximum latency constraints). Achieved results for the proposal are compared to those of some baseline solutions. Finally, the main open issues are discussed in order to provide a starting point for the following chapters of the thesis.
- **Chapter 4**: future 6G networks will leverage on the complementarity of edge computing and machine learning (ML) to develop an intelligent edge, where communication and computing resources will be collaboratively orchestrated. This chapter includes the second contribution of the thesis, thus the implementation of a set of ML algorithms to judiciously orchestrate the placement of delay-constrained computing tasks in the softwarized edge domain. Extensive simulation results are given, to demonstrate how well the methodologies under consideration perform in terms of model accuracy, complexity, and network-related variables.
- **Chapter 5**: in this chapter is tackled the design and implementation of an heuristic algorithm that solves the task placement problem in a feasible and efficient manner. Moreover, as a step further, is formulated a new optimization problem which takes into account the *compute reuse* of the computed tasks' output. This concept brings several benefits such as avoiding redundant computations and data exchange, reducing service provisioning time and enhancing edge resource utilization. A literature overview about this topic is first provided. The designed heuristic algorithm is validated against the solution achieved through a standard optimization solver and another benchmark heuristic in the literature. Performance is evaluated under differ simulation settings.
- **Conclusions and research open roads**: Finally, this chapter will conclude the thesis by summarizing the main results of the conducted study and identifying some possible future research directions.

2

Background and motivations

In this chapter the background of the thesis will be presented, by describing cloud computing and its evolution towards edge computing and edge intelligence.

2.1 Cloud computing: an overview

The huge amount of data generated by the billions of devices connected to the network raised several demands in terms of processing and storage capabilities. Cloud computing was the one of the first answers to such demands. It is a paradigm that enables the access to a set of shared and configurable resources (e.g., networks, storage, applications and services, servers), which can be rapidly provisioned through interaction with *service providers*. It enables users to take advantage of remote infrastructure, platforms and software, meeting the needs for processing and storage resources required by today's applications, which are not available locally on devices. There are numerous actors involved in a cloud computing system, in the fruition and delivery of services. Fig. 2.1 depicts the main actors in cloud computing:

- *Cloud Consumer*: person/organization using the services offered by the Cloud Provider;
- *Cloud Provider*: organization/entity that provides the services;
- *Cloud Auditor*: third party who monitors the services, their performance, security and implementation;
- *Cloud Broker*: entity that manages the use, performance and delivery of cloud services and negotiates the relationship between the Cloud Provider and the Cloud Consumer;
- *Cloud Carrier*: intermediary that provides connectivity between Cloud Provider and Cloud Consumer.

The services that the *Cloud Provider* offers are grouped into three categories (Fig. 2.2) that differ in the level of access to the Cloud system provided to the consumer:

1. **Infrastracture as a service (IaaS)**: is a service model in which the user is provided with resources, such as: storage, computational capacity, and network access. The *Cloud Provider* manages the entire physical infrastructure that the user can develop by

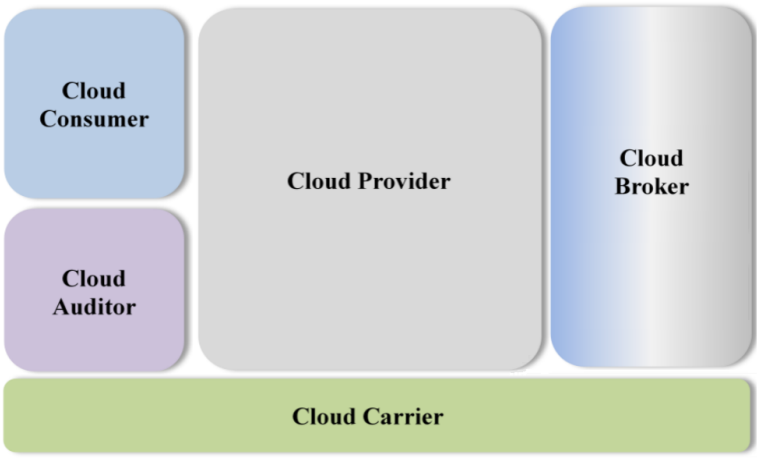


Fig. 2.1. Architecture of a cloud computing system and key players [2]

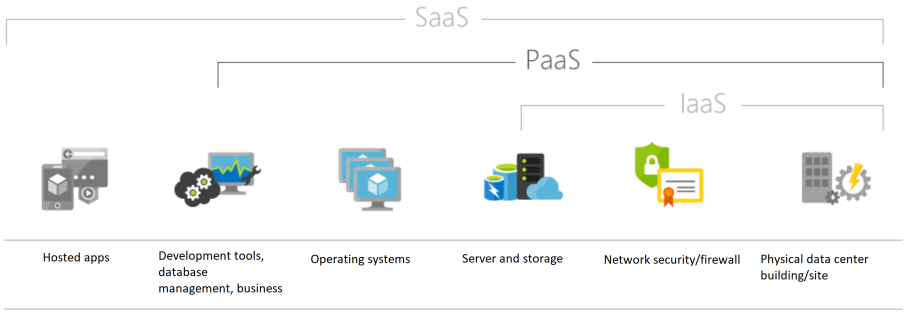


Fig. 2.2. Cloud service models [3]

inserting operating system, applications and programs, and ways to interact with the infrastructure;

2. **Platform as a service (PaaS)**: is a model in which the *Cloud provider* manages the infrastructure and development environment, operating system, programming language, libraries, and services that will be used. The *Cloud Consumer* has, therefore, a platform with which he/she is able to develop, implement and manage applications with only the programming languages provided by the provider.
3. **Software as a service (SaaS)**: is a model according to which the provider offers to the user an operational structure, in which he or she can use programs, which can be accessed diversely with various devices through appropriate interfaces. The user can only change settings at the application level, but cannot make changes in the basic structure of the cloud system.

2.2 From cloud computing to edge computing

The following section will present the main features of the Mobile Cloud Computing paradigm and its evolution toward Mobile Cloud Computing and Multi-access Edge Computing

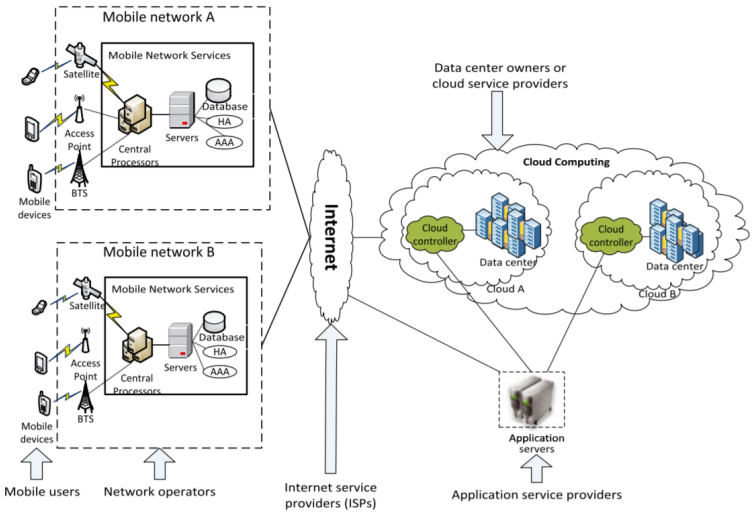


Fig. 2.3. Architecture of Mobile Cloud Computing [4]

(MEC). The latter moves the execution of services as close as possible to the user, producing several benefits for both the network and the user. These features make the MEC paradigm a key technology for fifth-generation and future systems, ensuring low latency in accessing computation and storage resources.

2.2.1 Mobile Cloud Computing

The growth of Cloud computing and the emergence of new mobile applications paved the way to the development of the Mobile Cloud Computing (MCC) paradigm. Thanks to the resources offered by the cloud, the users equipped with mobile devices can enjoy services that they would not be able to support locally because of their constrained capabilities. The MCC provides new Information Technology (IT) applications not only for the smartphones, but also to a plethora of mobile devices that need high computing and storage capabilities [4], such as healthcare devices or drones.

Therefore, the MCC paradigm is apt to empower mobile devices through the fruition of services implemented on remote data centers to run a wide set of applications with high complexity, think about gaming, image and video processing, etc.

Bringing the execution and storage of data to the Cloud enables the fruition of services without burdening mobile devices and without exhausting their typically limited local resources (memory, CPU and battery).

An example of a Mobile Cloud Computing architecture is shown in Fig. 2.3. It can be seen that there is the presence, on the Cloud side, of a *cloud controller*: the entity to which requests and information from users connected to the *Internet* are transmitted over heterogeneous networks. In fact, *mobile devices* can be connected to the network through different radio technologies: satellite networks, Wi-Fi access points, cellular networks (3G, 4G, 5G), *Wireless*

Sensor Network (WSN), mesh networks. Each *cloud controller* is associated with a cloud *data center* that responds to service requests from users.

Mobile devices access remote cloud services through network connectivity provided by *mobile network operators*, which can provide services such as authentication, authorization and access according to their own policies by consulting their *databases*.

The MCC brings several benefits and drawbacks.First of all, executing the intensive tasks outside the user devices allows to save battery capacities in case the energy cost to send and to receive the data is less than the energy cost to process the tasks into the device. Thanks to the higher processing capabilities rather than the mobile devices, the execution delays are lower. Moreover, the cloud data centers have higher storage capabilities than users' devices. Finally, with MCC the performances, resources, and functionality can be increased or decreased based on need or specific requirements, thereby introducing greater scalability. However, MCC increases the usage of wireless network bandwidth. Furthermore, if not well managed, it could introduce vulnerability and privacy concerns. Since cloud service draws its foundation from access to resources via the network, it is essential to ensure not only connectivity, but also high upload speed. Hence the availability has to be guaranteed. Some cloud services, such as compute instances, have multiple subscription tiers and pricing schemes. These variables make pricing analysis difficult. Among the MCC-related issues, still open, there is the necessity to understand when and where to migrate the execution of a service as well as how much and what portion of data to be stored into the cloud. Migration can be triggered either to better follow the end user or for load balancing purposes. Processing and storing everything into the cloud it is not always the best solution. Indeed, for the next 5G/6G applications, it cannot be done as it would bring to a massive network congestion and a high computation delays because of the distance between users and the remote cloud.

2.2.2 Toward edge computing

In order to solve the aforementioned issues, computation and storage resources should be positioned as close to the mobile end devices as possible, for example, by putting cloud-like servers inside cellular Base Stations (BSs) or Access points (APs) depending on the network architecture.

The European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) first coined the term *Mobile Edge Computing* (MEC) to describe this trend of placing cloud capabilities close to mobile end devices at the Radio Access Network (RAN) premises. Mobile Edge Computing was renamed to Multi-acces Edge Computing (MEC) by ETSI ISG in September 2016 in order to expand the utilization of MEC to heterogeneous networks technologies, such as WiFi and fixed access, besides cellular networks (see Fig. 2.4).

There are numerous motivations behind the evolution of cloud computing toward MEC.

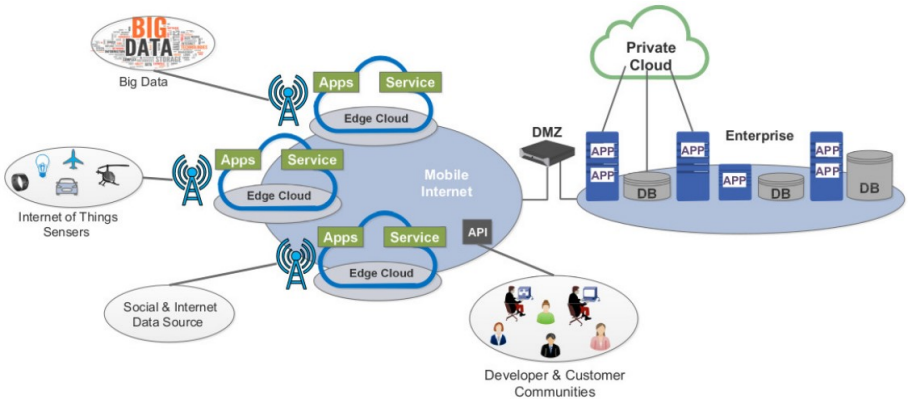


Fig. 2.4. Placement of MEC servers on the mobile access network [5]

One of the main factors contributing to the growth of IP traffic is the consumption of video, often in high definition (Ultra-high definition (UHD), 4K), due to the growing role that content delivery networks (CDNs) are playing.

Moreover, new applications such as Augmented Reality (AR), which are computation-intensive, and delay and jitter-sensitive, are growing now. Video traffic as well as AR requires to be enjoyed with low latency and high throughput and can benefit from storage and processing resources close to the users. Thus, this approach would, on the one hand, enable higher Quality of Experience (QoE) for users who require speeds in the order of hundreds of Mbps, and on the other hand, it would enable a decrease in congestion in the core network because requests for the same content from multiple users are fulfilled by the access network without the need to establish connections with remote servers.

In the Internet of Things (IoT) sector, MEC could also play a major role. In the IoT world, objects gain intelligence by being able to communicate data about themselves and what they measure. Typically, an IoT object has limited resources in terms of computational capacity and memory. It is the gateway node that stores and aggregates data from IoT sensors/actuators. This type of traffic may require very low latency and need a large amount of memory to provide efficient real-time services. In addition, being generated by massively deployed devices it may overwhelm the network to reach the remote cloud.

In addition, the dominance of smartphones, as a communication tool (social media, video consumption, IoT application monitoring, video surveillance, gaming) and the rise of other mobile devices prompted the industry and standardization bodies to evolve Cloud Computing toward a solution that could decrease the load on the core network and latency, bringing Cloud services closer to the users.

Thus MEC was born, in which computational and storage resources are moved closer to the end user thus decreasing latency. However, moving to the edge of the network results in fewer such resources than those available in remote data centers and it is not free of issues.

In addition to MEC technology, there are other proposals in the literature that enable the processing of requests at the edge of the network, such as Fog Computing and Cloudlet. A comparison of the three technologies of the so-called *Edge Computing* is made in [12].

2.3 The ETSI MEC architecture

According to the ETSI specifications [6], the MEC paradigm allows applications to be implemented as software entities running on a virtualization infrastructure that allows them to be decoupled from the server hardware that hosts them. MEC servers are the nodes that host computing, storage and network resources and are located close to access networks. They can be located, for example, near Sink nodes, Wi-Fi APs, eNodeBs, Radio Network Controllers (RNCs).

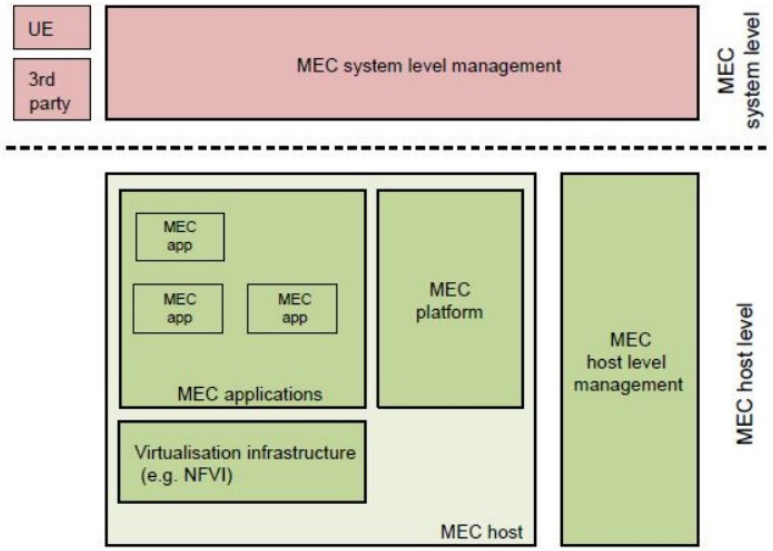


Fig. 2.5. ETSI MEC architecture [6].

The system level and host level components are highlighted in the MEC reference architecture (seen in 2.5) whereas the network level is hidden because there are no MEC-specified reference points to access those entities.

The MEC host is a logical construct that incorporates both the MEC platform and the virtualization infrastructure. It gives the MEC applications access to compute, storage, and network resources. A data plane component of the virtualized infrastructure is responsible for enforcing the forwarding rules that the MEC platform receives and for routing traffic between the applications, services, and networks.

On top of the virtualization infrastructure that the MEC host provides, MEC applications are executed as virtual machines. To manage the MEC services offered in that MEC host, the applications communicate with the MEC platform.

The system level management evaluates the service and resource needs that the MEC application may declare at the time of instantiation, for example, its limitations on the maximum permissible latency. Based on these requirements, the decision to move applications and the choice of the target MEC host(s) are made.

The MEC platform consists of a set of fundamental features that are needed to run applications on the MEC host and make it possible for MEC apps to find, promote, offer, and use MEC services. These fundamental features include time references, persistent storage, and traffic directing.

The MEC platform also supports setting up a local Domain Name System (DNS) proxy or server to route user traffic to the MEC applications.

The MEC host level management includes the MEC platform manager and the Virtualisation Infrastructure Manager (VIM)

The MEC platform manager consists of the MEC application lifecycle management (LCM), MEC application policy management, and MEC platform element management functionalities, all at the host level.

A MEC application is instantiated, terminated, and relocated by the application LCM, who is also in charge of alerting the MEC orchestrator to certain application-related events. Authorizations, traffic regulations, DNS configurations, and problem-solving when a group of policies are in conflict are all included in policy management.

In order to manage virtualized resources for MEC applications, such as allocating and releasing virtualized compute, storage, and network resources, the Virtualization Infrastructure Manager (VIM) communicate with the Virtualization Infrastructure of the MEC host.

The MEC system level management has a pivotal part in MEC architecture as it contains the MEC orchestrator and the Operations Support System (OSS). Due to its insight into all of the MEC system's resources and capabilities, the MEC orchestrator plays a crucial role[1]. It is responsible for a variety of tasks, including resource management, instantiation coordination, healing, and conflict resolution.

Moreover, the MEC Orchestrator is accountable for managing the MEC applications and the associated processes by assisting with application onboarding, verifying their integrity and authenticity, validating the policies attached to them, and keeping a catalog of the available applications.

By choosing the suitable target MEC host and, if necessary, initiating the application relocation, the MEC orchestrator is responsible for ensuring that the application's requirements (such as latency, user throughput, etc.) are met.

The highest level management system to help get the MEC applications running in the appropriate location of the network is the operator's Operations Support System (OSS).

[1] It is worth to notice that the allocation strategies proposed in the next chapters of the thesis can be also deployed as MEC services on top of the (MEC) orchestrator, within the ETSI MEC architecture

The clients in the user equipment (UE) and third parties send requests to OSS, which collaborates with the MEC orchestrator to instantiate and terminate the MEC apps. Third parties access to MEC orchestrator thanks to a Customer Facing Service portal (CFS).

2.4 Benefits

As mentioned in the introduction of this chapter, over the past decade, cloud computing has catalyzed a great deal of interest in the context of enterprise IT infrastructure, since it offers more advantageous solutions to meet the increase in processing and storage demands.

However, there has been an increase in the number of devices, other than typical servers or PCs, whose functionality cannot be increased by the cloud alone. The great influence of IoT, which has initiated the development of wearable devices, *smart environments* and, in general, devices that can connect to the Internet without user interaction, has changed the scenario of devices used at the edge of the network.

As a result of these trends, ETSI developed the MEC architecture described in the previous section to meet the above needs.

The main benefits that a MEC solution can offer are as follows:

1. *QoS with real-time requirements and lower latencies*: the latest generation of mobile devices have an increasingly high QoS demands due to their mobility, and the stringent requirements of real-time and interactive applications. Cloud computing alone cannot be the optimal solution to these demands, since packets must traverse many nodes before accessing the networks where remote servers are located. The MEC approach allows the latency experienced to access cloud services to decrease.
2. *Increase of Battery life*: when it comes to mobile devices, one of the concerns to face is battery life. One of the benefits of MEC is precisely that it runs tasks on the edge infrastructure instead of the device itself. Cloud computing provides the same benefit, but requires more battery consumption than MEC because of the data transmission costs. Indeed, the probability of packet loss over a longer route is greater; such loss results in more retransmissions resulting in higher battery consumption [13].
3. *Lower congestion in the core network*: providing a service as close as possible to the user decreases the overall network congestion because input data do not need to traverse the network. Moreover, performing storage of a data item (e.g., high-definition multimedia content) in MEC servers is beneficial for the network in the case where the required content is popular content because it avoids the transmission of large identical packets occupying a lot of bandwidth in the core network.
4. *Scalability*: deploying services and applications by replicating them in the form of virtual machines creates an opportunity to make network management more scalable, even in the face of the large traffic generated by IoT devices;

5. *Resilience*: running applications and services at the edge provides the benefit that, in the event of anomalies or errors, the problem does not affect the entire network and this can be addressed more easily;

2.5 Use cases

ETSI defines several use cases that can benefit from MEC and can be broken down into the following services' categories [14]:

1. *The consumer-oriented services*: are the ones that improve the end-user experience directly. Indeed, these services are considered too computationally intensive to be executed by the User Equipment (UE) and too latency-sensitive for execution in the cloud. Some example are gaming, remote desktop applications, and augmented and assisted reality.
2. *The operator and third-party services*: use the MEC infrastructure to create services that are not directly aimed at the end-user. They offer services to the applications that are end-user-oriented. These services can take advantage of the low latency of MEC infrastructure, but also of the redundancy reduction of generating a single service that can serve several users. Some example are active device location tracking, big data, and enterprise services.
3. *The network performance and QoE enhancements-oriented services*: are oriented to enhance the network and the QoE without offering new applications or services to end-users. These services can reduce the costs of the Mobile Network Operators (MNOs) while improving the network's efficiency. Some example are content/DNS caching, and video optimization;

Some of the 36 use cases provided by ETSI in [14] are described below. They are the ones closest to those considered within this thesis.

2.5.1 Use Case 1: Local content caching at the mobile edge

The use of mobile broadband services has significantly increased as a result of location-based services. The graphic and display processing technology has advanced significantly, and portable devices can now play high-resolution video. Additionally, widespread use of social media enables quick and effective sharing of themes that first emerge in viral style.

The content is frequently consumed at around the same time and in the same location due to viral propagation. As a result, there is more pressure to provide enough bandwidth, and typically the mobile broadband network's capacity turns into a bottleneck. The solution to this issue is local content caching, which can reduce costs associated with backhaul and transit while also enhancing customer QoE. The backhaul capacity needs could be cut by up to 35% using content caching [14]. When asked, a MEC application can serve requests from the local cache by storing the most popular content that is consumed locally. In such instance,

there would be no need to transport contents via the main network, which would result in huge backhaul capacity savings. Additionally to capacity reductions, download times for the content can be significantly shortened.

2.5.2 Use Case 2: Augmented reality, assisted reality, virtual reality, cognitive assistance

By doing an analysis of their surroundings, determining the semantics of the scene, augmenting it with additional knowledge provided by databases, and feeding it back to the user within a relatively short period, Augmented Reality (AR) enables users to enhance the experience about the environment they are immersed to. A smartphone or any wearable equipment with a camera and other sensors can be taken as an example of a device which can exploit an AR service. Cognitive assistance advances the idea of AR by giving the user tailored feedback on tasks they may be carrying out (e.g. cooking, recreational activities, furniture assembling, etc.). In a relatively short period of time, the scene analysis and user recommendations must be provided. Low latency applications, such as games, AR, or Virtual Reality (VR) apps, have the option of implementing the rendering pipeline either directly on the client device or in a MEC application on the MEC host. These apps can decide to choose a MEC application running on a MEC host to offload a portion of the device's computing load. This might comprise artificial intelligence, simulations of physical laws, and other elements.

2.5.3 Use Case 3: Gaming and low latency cloud applications

Games are a fairly common application on laptops, tablets, and mobile devices. However, because the servers are frequently accessible through the Internet and are placed outside of the RAN and Core Network, many games played on computers connected via LAN and/or broadband Internet connection demand low latency values that are typically not currently attainable for UEs. Users will get access to a new class of low latency-based games by placing game server applications closer to the RAN. Naturally, the use is not limited to games and can be advantageous for any form of low-latency applications, such as employing a "remote desktop" protocol to access cloud virtual machines when the computing capacity is too big to be effectively operated on a tablet. In this instance, a very small latency between the user's activity and the device's feedback is required for the user experience to be satisfactory.

2.5.4 Use case 4: Automotive applications

By exchanging safety-critical data, communication between vehicles and roadside sensors and a roadside unit aims to improve the safety, effectiveness, and convenience of the transportation system. The roadside application uses algorithms to identify high-risk situations in advance and notify and warn nearby cars by using information from vehicles and roadside sensors. The drivers of the cars can respond right away, for instance by avoiding the lane

danger, reducing speed, or altering the course. In some situations, communication between vehicles and infrastructure must have latency requirements as low as 10 ms. A nationwide Digital Short-Range Communications (DSRC) network would not be required because messages, such as hazard warnings (such as accidents, danger on lanes, etc.), may be transmitted in real time through Long Term Evolution (LTE). The LTE technology as well as 5G could supplement DSRC in installations where it is present. The connected automobile cloud can be expanded into the highly dispersed mobile network environment via multi-access edge computing. The roadside functionality can be provided by applications that can be installed on MEC hosts.

2.5.5 Use case 5: Industry 4.0

Sensor networks in a plant use a variety of wired and wireless sensors, including microphones, CO2 sensors, pressure sensors, humidity sensors, and thermometers, to monitor processes and the relevant parameters in an industrial setting. By utilizing machine learning (ML) methods, the monitored data from such a system is used to find its anomalies. Before an algorithm may later operate on a portion of the available measured data, it has to be trained. Both the training and the inference phase (in this case anomalies findings) can be carried out centrally or decentralized. MEC might be utilized to host a local monitoring function, which could give the basic sensors access to additional external computational resources. In order to maintain sensitive data in a fabrication site and keep the automated process independent of an Internet connection, such a local method is chosen over a more centralized one. Furthermore, industrial and intra-logistics contexts often demand for mobile robots and mobile platforms, such as automated guided vehicles (AGVs). A guidance control system monitors and manages mobile robots and AGVs. Such a remote control system may be hosted by MEC and would, for instance, support the processes for handling materials and commodities, particularly incoming and outgoing materials, in warehousing and commissioning, in transportation as well as the transfer and provision of items.

2.5.6 Use case 7: Compute-intensive application offloading

The MEC host executes *compute-intensive* tasks with high performance in the application computation offloading use-case rather than a mobile device. The user experience can be met regardless of the kind of UE by offering extensive computation resources on a MEC host, which enables application computation to be offloaded and accelerated even when a user utilizes relatively low performance devices. In particular computation-intensive applications like graphic rendering (high-speed browsers, virtual reality, 3D games, etc.), intermediate data processing (sensor data cleaning, video analysis, etc.), and value-added services, this use-case works well (translation, log analytics, etc.).

2.6 Edge Intelligence: the interplay of edge computing and artificial intelligence

Several emerging and promising technologies will increasingly be incorporated into edge computing it as it advances. Among them, the fusion of edge computing and Artificial Intelligence (AI), referred to as edge intelligence (EI), is a key development path and offers a great opportunity for growth. In order to support the performance, new unprecedented functionalities, and new challenging and demanding services of future 6G networks, edge intelligence (EI), driven by AI techniques (e.g., machine learning, deep neural networks, etc.), is already regarded as one of the main missing components in 5G networks [15].

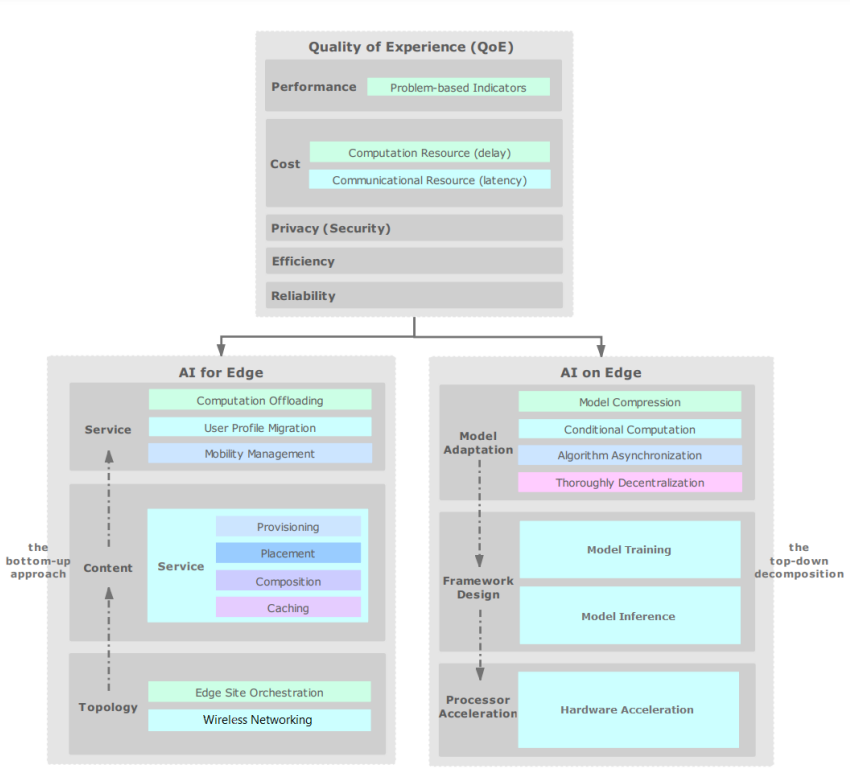


Fig. 2.6. The research roadmap of Edge Intellegence [7].

Fig. 2.6 shows the research roadmap of edge intelligence presented in [7], although it is still in its early stages. It describes a logical separation between the two directions respectively, i.e., *AI for edge* (left) and *AI on edge*, a.k.a. *edge for AI*, (right). The key priority on the roadmap is QoE (on the top of the figure), and every implementation and proposal belonging the two main research directions have to guarantee the QoE that is determined by jointly considering multi-criteria: performance, cost, privacy (Security), efficiency and reliability.

In AI for Edge the performances are problem-dependent. For instance, performance indicators might be the ratio of successfully offloading in the case of a computation offloading problem.

Cost is typically made up of communication and computation costs. While communication cost displays the demand for communication resources such as power, frequency band, and access time, computation cost indicates the need for computing resources such as accomplished CPU cycle frequency and allocated CPU time.

According to the AI for edge research's direction, AI offers effective tools for tackling challenging learning, planning, and decision-making issues in the edge computing domain. For example, research on the automation of unmanned aerial vehicles (UAVs) became quite popular with relation to the topology of an edge site. UAVs with a small server and an access point can be thought of as highly maneuverable moving edge servers [16]. Additionally, network planning researches effective middleware and protocol management. Intelligent networking, which entails constructing an intelligent wireless connection mechanism via well-known AI technologies, has been increasingly popular in recent years. AI can be used for example to address task offloading problem in a more dynamic and efficient way [17]. Moreover, it can face the service placement issue that is an important complement to service provisioning, which studies where and how to deploy complex services on possible edge sites [18]. Service composition investigates how to use AI technologies to choose potential services for composition in light of energy usage and mobile end users' quality of experience. As surveyed in [19], AI could be also used for *in-network caching*. The main use of AI in caching is to determine what and when to cache in order to achieve the best possible caching goals (cache hit ratios, latency, throughput, etc.). In other cases, it is possible to utilize a variety of machine learning (ML) approaches for wireless networks with the aim of finding answers to related issues including popularity prediction, cache decision optimization, and user clustering. Additionally, Computation offloading searches for AI solutions for the load balancing of different computational and communication resources with the goal of making edge server selection and frequency spectrum allocation. Finally, while for user profile migration AI research efforts are made in the case of mobile users' mobility to adapt the location of their profiles such as private data, logs, and configuration files, when it comes into mobility management AI is used for statistics and probability theory.

On the right of the Fig. 2.6, the research road of AI on edge is depicted. It examines how to use the network edge for AI models' training and inference processes. Indeed, the core of this research topic (framework design) has the objective to improve, respectively, the applicability of existing frameworks to the edge and the computation performances of the AI models operations. Without changing the current AI models, framework design seeks to provide a better training and inference architecture for the edge. New frameworks are being developed by researchers for both model training and model inference. Just to cite an example of a data splitting training framework, Fig. 2.7 shows a framework in which an integration of Federated Learning (FL) within Edge is designed to provide a better training

and inference architecture [8]. In the framework, each part of the Mobile Network Operator (MNO) core (green areas) has different AI clients which refer to a single FL central server. Then, with a hierarchical manner with respect to the processing capabilities of the nodes, the FL central servers become the clients of the a centralized FL central server, which aggregates each AI model of the entire network. Specifically, in order to train a general DL model, the FL central server iteratively asks a (random) sample of clients (distributed devices that train DL models) to 1) download model parameters from a particular server, 2) use the downloaded model to train with their own data, and 3) upload only the new model parameters to the server, which aggregates uploaded updates from the clients to further improve the model. In conclusion, FL keeps the training data local while allowing resource-constrained edge computing devices (including UEs and edge nodes) to develop a shared model.

Based on current training and inference frameworks, model adaptation makes the necessary changes. Model compression, conditional computation, algorithm asynchronization, and thorough decentralization are only a few techniques for achieving model adaption.

In conclusion, processor accelaration includes all hardware changes that are made to speed up the execution of AI algorithms.

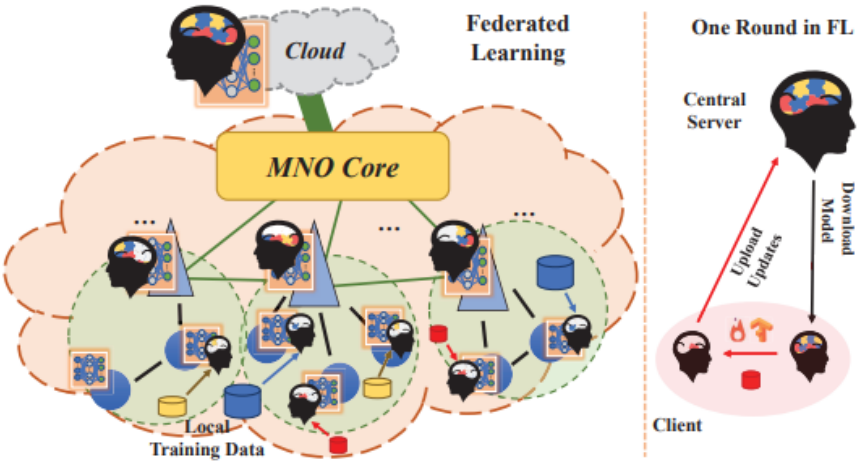


Fig. 2.7. Deep Reinforcement Learning (DRL) with Federated Learning framework [8].

2.7 Open issues

MEC practical projects currently require improvements that can considerably improve the user experience and the costs for MNOs while being an expanding research subject with a variety of commercial and business objectives.

Despite the clear benefits of MEC, several issues lie ahead to make the best of this paradigm and deserve further investigation.

Mobility awareness, offloading decision, and privacy are listed as some of the open issues identified in ETSI specifications [14, 20, 21]. Within the thesis work, the challenge of task offloading decision has been faced, and the rest of the thesis is focused on this aspect.

Thus, in the following of this chapter open relevant open issues are discussed and some main questions are made to start from them for future works that are beyond the scope of this thesis work.

2.7.1 Mobility

The gaps to support mobility that are not covered by existing works in the context of MEC are identified in [20]. The distance between a UE and the MEC host it is serving is strongly connected with the experienced MEC QoS.

When a UE asks for a MEC host's assistance, the orchestrator can select the best MEC host to provide that assistance and launch an instance of the MEC application there. But as a UE moves, it can recede from the MEC host that is serving it and it can approach to others MEC host in the network, which would reduce the QoS, e.g., the latency constraint cannot be met due to the larger distance. Mobility may also cause the UE to approach more MEC hosts, switching the position of the ideal MEC host to the new one. When this occurs, the MEC system may cause the migration of an instance of a MEC application from the previous optimal MEC host to the present optimal one. By transferring the application context and adjusting the network's traffic forwarding settings in this case, the MEC system instantiates the MEC application in the new MEC host.

We can list some of the open topics pertaining to mobility as follows:

1. How can the migration overhead be reduced?
2. Which is the most appropriate moment for application migration?
3. Which part of the application exactly should be moved?

2.7.2 Security concerns

The white paper in [21] explores security-related use cases and requirements and aims at identifying aspects of security where the nature of edge computing results in insufficient industry approaches to cloud security. Since cloud computing involves delivering data and apps to a third party's custody, privacy is a common worry [22]. MEC raises problems that are comparable but not exactly the same. Some privacy-related MEC open issues are reported below:

1. How may various access networks, such as WiFi, LTE, and 5G, affect consumers' privacy?
2. In comparison to the cloud, how susceptible is MEC?
3. To what extent are MEC hosts varied in terms of privacy vulnerability?

3

Task allocation at the edge

In this chapter, the task allocation challenge at the edge is discussed. Firstly, focus will be on the main context and the state of the art of this topic, then the first contribution of the thesis is presented. The following sections will provide the performance evaluation of the designed solution and finally the main concerns are discussed in order to provide a starting point for the following chapters of the thesis.

3.1 Main context

Task offloading to the edge has drawn a lot of research attention thanks to advancements in virtualization and networking technologies, as thoroughly assessed in [23]. Applications like AR, online games, and autonomous driving (see section 2.5), which rely on resource-intensive tasks (e.g., ML inference algorithms for object detection) [24], may be offloaded to the edge. Fig. 3.1 shows an user equipment offloading request to the base station is it connected to. The base station is enhanced with the storage and computing capabilities of the edge node. These applications hardly run into low-resource mobile devices and for them the response time is critical to ensure the user experience.

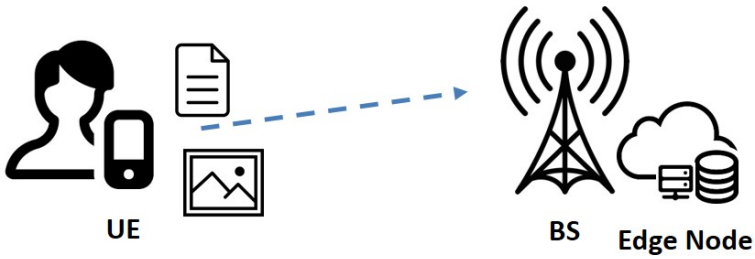


Fig. 3.1. Task offloading at the edge.

In order to offload a computing task to the edge, *(i)* a node must be chosen as the task executor that has the necessary processing resources, and *(ii)* input data must be moved from the source to the chosen node for processing.

The executor can be either the remote cloud in compliance with a conventional cloud-based solution or an edge node among the multiple ones in the edge infrastructure.

A proper orchestration of the available edge resources is required to choose *where* to execute computing tasks due to the time-varying availability of computing, storage, and network edge resources, the limited edge nodes' capabilities compared to the remote cloud, and the peculiarities of the aforementioned types of intelligent applications. This is significantly more difficult when edge nodes are not limited to purpose-built servers, but the edge computing domain includes numerous network nodes connected to one another and with largely heterogeneous capabilities, such as those making up a campus network or the backhaul portion of a mobile network. Such a trend is fueled by recent initiatives pushing *in-network computing* like Internet Engineering Task Force (IETF) Computing in the Network (COIN) Research Group [25], [26] and by recent research works [27], [28].

In the scientific literature, offloading strategies are suggested to reduce energy consumption, meet delay requirements, and preserve network bandwidth [29], [30]. Generally, the edge node nearest to the data sources is the best location for carrying out an offloaded task. Such a decision often decreases the task executor's response time and conserves network bandwidth for input data delivery. Although edge nodes have limited computing resources, outsourcing all computing tasks to the nearest edge node may result in performance degradation if the processing workload is too high. This is because of the wait time before serving the computation request as well as the execution time itself. Furthermore, given the very delay-sensitive tasks anticipated in sixth generation (6G) systems [31], this may not be feasible.

3.2 Contributions

To face the orchestration issue already described in the above section, we provide a new solution for allocating computing tasks in the edge that minimizes the network's resource consumption and limits the task execution latency at the edge node serving as the task executor.

In this chapter, it is described an initial task placement proposal we published in [32] that will be improved within the next chapters. Without loss of generality, we refer to an SDN-based method to implement the intended centralized task placement strategy for each solution described in the thesis. Such a decision facilitates the joint optimization of network and compute edge resources and is in line with recent literature aiming for a unified network-edge service provisioning [33].

3.3 SDN

3.3.1 SDN: basics

SDN is an emerging network architecture where network control is decoupled from forwarding and is directly programmable [9]. Fig. 3.2 shows the SDN architecture. By separating

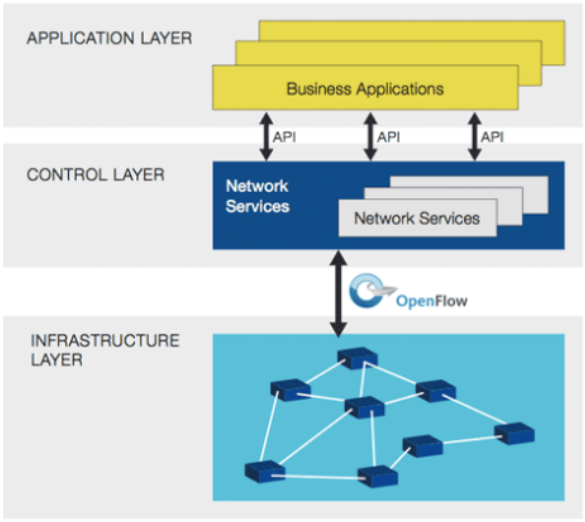


Fig. 3.2. SDN Architecture [9].

the control plane from the data plane and transferring it to a conceptually central location, the Controller, SDN revolutionizes networking. The network applications (such as load balancing and routing) are applied on top of the underlying network nodes as straightforward forwarding components. The Controller maintains a network-wide perspective of the connection status and network nodes under its control, which makes it easier to design sophisticated mechanisms for resource management and traffic control. This is made possible by adding the appropriate rules to the flow table of the nodes under the Controller's control thanks to the first established standard communication interface, called OpenFlow.

Business applications inside the application layer directly inform the SDN Controller about the desired network behavior and network requirements.

3.3.2 SDN-enabled task orchestration: literature overview

SDN has also been seen as a major enabler for task orchestration in edge computing scenarios [33].

Thanks to the decoupling of data plane and forwarding plane, thus having the SDN controller a centralized awareness of the system, and thanks to its traffic engineering capabilities, SDN technology plays a crucial role for the joint optimization of network and computing resources at the edge [28]. In [34], a task offloading method in a software-defined ultra-dense

cellular network where BSs are equipped with edge computing capabilities is provided. An SDN Controller informs the mobile devices on whether to offload a task or execute it locally and, in the former case, to which base station, in order to reduce task completion time, by taking into account the network status from a global perspective. A task allocation SDN edge framework for vehicle networks is put forth in [35] with similar objectives. There, the SDN Controller gathers task data from cars and monitors the condition of distributed edge servers, including how much memory and CPU are available. The Controller advises the cars to perform computation tasks locally or to offload them to a nearby edge server based on this information. A vehicle mobility analyzer at the Controller is used in the same scenario by the work in [36] to forecast the communication time between the vehicle and the close-by edge servers. The task is then proposed to be offloaded to the *best edge server*, i.e., the one that increases the likelihood that the entire task will be executed successfully within a maximum completion time.

The emphasis in [37] is on computing workloads that call for numerous input data from heterogeneous end devices in an edge mesh scenario, i.e., a wirelessly connected collaborative edge network. When transmitting the input data, the authors take traffic congestion and network bandwidth consumption into account. They also research the task allocation problem to simultaneously schedule tasks and network flows with the goal of reducing service completion time. It is suggested to use a multistage greedy adjustment algorithm, implemented at the SDN Controller, to support task allocation in accordance with the bandwidth of the flows.

In [38], an SDN-based architecture is presented to position edge clouds on access points in the best possible way and to schedule computing tasks while assuring the least amount of overall energy consumption while being compliant with task delay requirements.

3.4 System model

In this section, the reference scenario and the main problem assumptions are firstly made. Then, there is a focus on the assumptions related to the computation delay model.

3.4.1 Reference scenario and main assumptions

As a reference scenario for our research, we consider an edge network domain overseen by an SDN Controller, as shown in Fig. 3.3. The domain is made up of a collection of N wired interconnected edge nodes that may communicate with the Controller directly thanks to SDN. Such nodes may include access points, base stations, and edge routers, as is typical in a mobile network operator's backhaul network. The computing capacities of each node i (with $i \in N$) are expressed as μ_i in terms of CPU cycles per second.

Executors of tasks can be edge nodes. Additionally, computing tasks can be carried out in the distant cloud whenever placing them at the edge is impractical due to the constrained

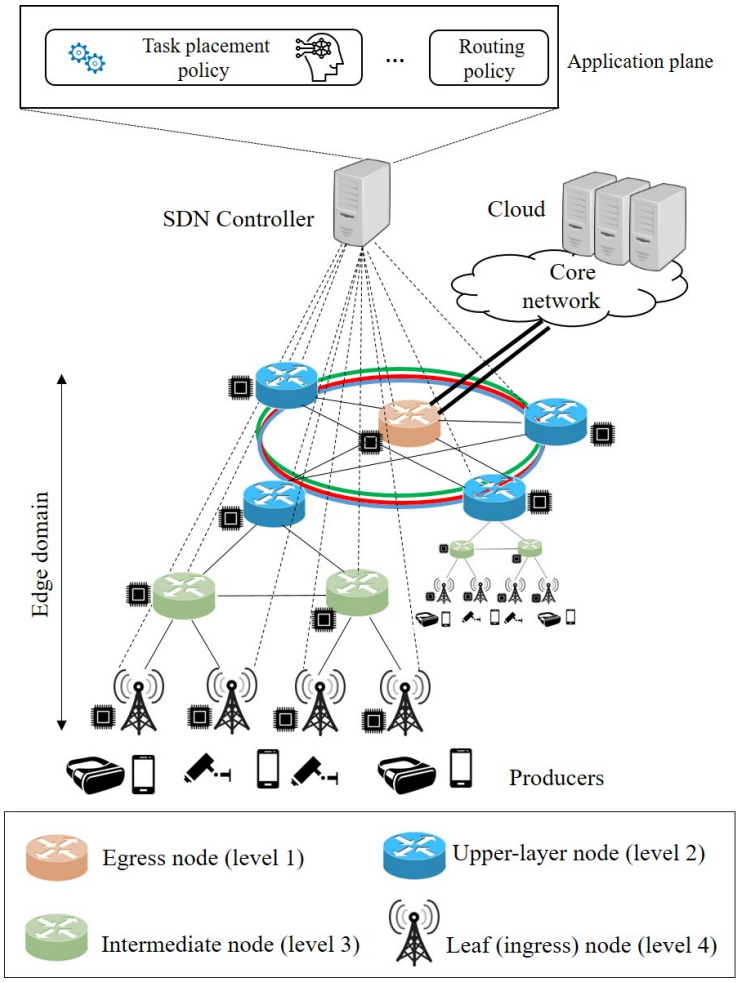


Fig. 3.3. Reference Scenario.

computing power of the edge nodes. For ease of representation, the cloud is shown as a single network node d with plenty of processing power. Some SDN nodes operate as ingress nodes, offering access to the domain for a set of *(i)* data providers, that create the input data for the computing tasks, and *(ii)* consumers, which ask for computing a task. When a computation over self-generated content is requested, these latter parties can likewise serve as providers. A core network segment and an egress node link the edge domain to the cloud.

The SDN Controller coordinates both the placement of computing tasks and the routing operations. Due to its native programmability, the SDN Controller in particular implements the deployment of computing tasks as a new network application. The built-in SDN routines provide a domain-wide view of the resources that can help the strategy. In fact, by utilizing the OpenFlow (OF) messages extension [28], the Controller can monitor the state of resources.

Despite the fact that we are considering a Controller with SDN capabilities, it is important to note that the developed task placement method may be deployed without SDN and is applicable to several centrally-managed architectures. It can be set up, for example, on

top of a MEC orchestrator, inside the ETSI MEC architecture 2.5, or on a specially designed proxy server. The latter can then communicate with a separate SDN Controller to obtain network-related metrics or impose extra functions (native to SDN) to obtain routing information, thereby achieving the stated goals of minimizing network resource utilization and adhering to task delay constraints.

A computing task, j, (with $j \in M$, being M the set of tasks), can be described as follows: j=$\{s_j, D_j^{max}, l_j\}$. Here, s_j denotes the size of the input data for calculation, D_j^{max} denotes the task's (j) maximum delay constraint, and l_j denotes the quantity of computing resources—measured in CPU cycles—necessary to complete task j. The key notations used in the thesis are summarized in Table 3.1, however when it will be necessary for the sake of readability, it will be updated with new notations for the next chapters.

In this investigation, the following fundamental hypotheses are done:

- Each computing task is assigned to a single edge node since they are *atomic*, meaning they cannot be divided into several subtasks.
- Tasks are *CPU-intensive* and *delay-constrained*. The first feature suggests that a given computing task will take significantly longer to execute than the amount of time asked to gather the input data at the task executor. The latter specifies a maximum delay within which the computing task shall be completed.
- The input data for each computing task may be retrieved by *multiple producers*. Hence, given a computing task j, we suppose that the total input data (of size s_j) consists of a collection of unique input contents of size s_{k_j} that are streamed to the executor from a variety of sources K_j. Therefore, it results $s_j = \sum_{k_j=1}^{K_j} s_{k_j}$.
- It is possible to ignore the amount of time required to send the computation's result from the edge to the consumer. In fact, the output of many applications, such as object recognition and tracking, is substantially less than the input data [39, 40].

Requests from consumers are routed through the edge node that the SDN Controller has designated as the executor before they are sent to the edge domain. The executor then contacts the producers to obtain the input contents, c_{k_j}. The output is delivered to the customer after the computation has been completed. The ingress node is instructed to forward the request to the cloud when a computation cannot be done at the edge.

It is worth to notice that the more efficient task placement made possible by the centralized approach comes at the expense of an unavoidable additional delay in task completion before the task allocation request reaches the Controller. To keep the ingress node-to-controller delay as short as possible, however, methods for the best Controller placement can be developed [41]. The literature frequently uses numbers in the range of 10 ms [41].

3.4.2 Computation delay

According to the relevant literature [38], [39], the computation delay is typically equal to the time required to process the task and is calculated as the fraction between the candidate

node's processing capacity and the computing resources required to complete the task. In the most common scenario, however, the computation delay of task j at edge node i includes two contributions, namely the *task execution delay* at node i and the *task queuing delay*, which is the amount of time spent at the same node before the task can be executed, in the event that node's resources are fully utilized because other tasks are already running on it. At each edge node, the following tasks are carried out sequentially: [42], [43]. Within the thesis, both latency contributions are taken into account.

Moreover, we assume that requests for computing task j arrive to node i at a rate that follows a Poisson distribution with parameter λ_j. In the related literature [30], [44], and [45], this is a widely held hypothesis. It is also justified by the statistics of Google data centers, which show that the arrival intervals between tasks are exponentially distributed [46], [47].

Additionally, we assume that the requested computing resources per task, l_j, are distributed exponentially with an average of $\bar{l}$. Therefore, to handle its corresponding computing tasks, each edge node can be modeled as an M/M/1 queue [48]. Hence, the corresponding service time follows an exponential distribution with parameter equals to $\frac{\bar{l}}{\mu_i}$.

The following equation describes the actual request arrival rate at a node i, which is still exponentially distributed:

$$\sum_{j \in M} X_{ij} \lambda_j, \quad \forall i \in N \cup d, \tag{3.1}$$

where X_{ij} denotes the binary variable:

$$X_{ij} = \begin{cases} 1, & \text{if the task } j \text{ is executed by the node } i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The average system delay of a M/M/1 queue, which includes the waiting delay and service time, is how we calculate the average computation delay for a generic task at node i:

$$\overline{D_i} = \frac{1}{\frac{\mu_i}{\bar{l}} - \sum_{j \in M} X_{ij} \lambda_j}, \quad \forall i \in N \cup d. \tag{3.3}$$

To keep the queue stable, the average arrival rate at node i, $\sum_{j \in M} X_{ij} \lambda_j$, should be smaller than the average service rate as expressed in Eq. (3.4).

$$\frac{\mu_i}{\bar{l}} - \sum_{j \in M} X_{ij} \lambda_j > 0, \quad \forall i \in N \cup d. \tag{3.4}$$

3.5 Problem formulation

Based on the conceived system model, the primary goals of our proposed task allocation strategy, DMEP, are to: *(i)* minimize the network resource utilization for data exchange; and *(ii)* meet the task delay requirements, in order to satisfy the end-users' QoS. The first objective, which precisely captures the operator's perspective, is part of the devised objective function. Additionally, the network operator wants to make sure that the delays of the

Table 3.1. Summary of the main notations.

Symbol	Description
N	set of edge nodes
M	set of tasks
i	generic edge node
d	remote cloud
μ_i	CPU capabilities of node i (CPU cycles/s)
j	generic computing task
D_j^{max}	maximum delay for task j to be executed
K_j	number of input contents for computing task j
s_j	size of the overall input data for computing task j
s_{k_j}	size of each input content c_{k_j} required for computing task j
l_j	CPU requirement for task j
X_{ij}	binary decision variable: 1 if task j is executed by node i, 0 otherwise
λ_j	arrival rate of requests for task j
Ω_{i,k_j}^{min}	number of hops separating the potential executor i and the closest providers for each input content c_{k_j} required for task j

computation tasks requested by the users is below a certain limit (D_j^{max}). In the following it is shown how a specific constraint in the specified optimization problem keeps track of this.

It makes sense that the edge node nearest to the source(s) of the input data would be the best location to carry out an offloaded task in order to reduce network resource utilization and conserve bandwidth. Despite this, edge nodes have limited computing resources. When all computing tasks are delegated to the nearest edge node(s), performance may suffer if the processing load is too high because of the delay before serving computation requests and the time it takes to actually perform the service.

Multiple paths could exist *between a given content source and a candidate executor*. Without losing generality, we assume that the Controller chooses the *shortest path* - that is, the one with the *minimum number of hops* - as the route by which input data are sent to the executor [1]. For a specific task, the shortest path is taken into account for each input content.

The minimum number of hops that the input data must traverse between the candidate executor, i, and each content source, k_j, in order to accomplish the computation task, j, is indicated by the symbol Ω_{i,k_j}^{min}.

We express the *network usage* (Eq. (3.5)) relevant to the execution of task j as the sum of the product of each input[2] content size (s_{k_j}) and the number of hops between the source

[1] The model could be enhanced in future works considering the bandwidth of the link

[2] If the exchanged output data is not negligible in comparison to the input data in terms of size, the model could be expanded to include it.

of the input content and the candidate executor (either an edge node or the cloud, hence belonging to the set $N \cup d$, henceforth denoted as $\widetilde{N}$), timed by the execution rate λ_j.

$$NU = \lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}, \quad i \in \tilde{N}; j \in M. \tag{3.5}$$

Therefore, the optimization problem can be described mathematically as follows:

$$\min \sum_{i \in \tilde{N}} \sum_{j \in M} X_{ij} \lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min} \tag{3.6}$$

s.t.

$$\sum_{i \in \widetilde{N}} X_{ij} = 1, \quad \forall j \in M; \tag{3.7}$$

$$X_{ij} \overline{D_i} \leq X_{ij} D_j^{max}, \quad \forall i \in \widetilde{N};\ \forall j \in M; \tag{3.8}$$

$$X_{ij} \in \{0, 1\}, \quad \forall i \in \widetilde{N},\ \forall j \in M. \tag{3.9}$$

Tasks are handled as atomic by the constraint in Eq. (3.7). In other words, each task is allocated to a single node, which accomplishes it in its entirety without being further divided into smaller tasks.

Every node is required to complete each task j within its maximum tolerable computation delay D_j^{max}, Eq. (constraint 3.8). Computing tasks will be transferred to the cloud when the edge nodes' low resources prevent them from ensuring the delay limitation Eq. (3.8). The cloud, on the other hand, can take advantage of higher, nearly unlimited compute capabilities.

We point out that nonlinear inequalities characterize this limitation, which is caused by Eq. (3.3). Finally, the restriction in Eq. (3.9) serves as a reminder that the computing task placement problem is most readily modelled using a binary integer variable.

The model can be formulated as a 0-1 ILP problem by considering the following reasoning: for each pair (i, j), if $X_{ij} = 0$ then the inequality in Eq. (3.8) is automatically verified, whilst if $X_{ij} = 1$ it is verified for $\overline{D_i} \leq D_j^{max}$. This enables Eq. (3.8) to be rewritten as the following linear inequality:

$$D_j^{max} / \overline{D_i} \geq X_{ij}, \tag{3.10}$$

that is Eq. (3.8) can be substituted by:

$$X_{ij} \leq D_j^{max} \left(\frac{\mu_i}{\overline{l}} - \sum_{j \in M} X_{ij} \lambda_j \right), \forall i \in \widetilde{N}; \forall j \in M. \tag{3.11}$$

3.6 Main settings

The conducted evaluation study aims to assess the performance of the proposed task allocation strategy. We compared DMEP with MEP, a less complex solution that merely seeks to reduce intra-domain traffic and ignores the delay constraint (in Eq. (3.8)). A *cloud-only placement* (CloudP) is also taken into account as a baseline approach.

Simulations have been conducted by using the standard solver provided through the integration of Matlab® and IBM CPLEX Optimization Studio tool on an Intel Core i7-6700HQ, 2.6 GHz (CPU), 16 GB (RAM), 512 GB (SSD), 1TB (HDD).

3.6.1 Network topology

We recreate in Matlab® a medium size network architecture with 29 edge nodes that simulates a metropolitan area network (MAN), similar to [28], and [49] . The roots of three-layered fat tree topologies are formed by four upper-layer nodes, that belong to the backbone ring and that are linked together in a full meshed topology. Providers and consumers are connected to the ingress nodes, or leaf nodes, at the base of each tree architecture. The egress node that connects the backbone ring to the cloud is a border router. Nodes have varying computing capacities (Table 3.2); in particular, it has been assumed for the sake of simplicity that an edge node's computational capacity increases with the distance from the provider/consumer towards the egress node [49]. It should be noted that the model as it has been defined is flexible enough to be used in other network contexts with different parameter choices.

3.6.2 Tasks settings

We take into account a set of 1000 computing tasks which differ with regard to the maximum computation delay and computation load. Table 3.2 provides a summary of all tasks' characteristics.

3.6.3 Metrics

The following metrics are analyzed:

- *Exchanged data*: it measures the network resource usage in the domain as the overall amount of input data (in bytes) transmitted by providers 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 amount of exchanged data.
- *Average queuing delay*: it calculates the average waiting period before tasks are carried out at the chosen edge node.
- *Average execution delay*: it calculates the average of the actual times required to process the task.
- *Average computation delay*: it measures the average computation delay, accounting for both the queuing delay and the execution delay.
- *Task offloading to the cloud*: it measures the percentage of requested computing tasks that have been transferred to the cloud.

Table 3.2. 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 [50]
Task catalog size	1000
Number of hops	• towards an executor in the edge [1,5] hops; [51], [52] • towards the cloud [14] hops.
Maximum computation delay (D_j^{max})	• uniformly distributed in [10, 100] ms [53], [54], [55] • uniformly distributed in [100, 500] ms
Average task computing workload ($\bar{l}$)	$10 \cdot 10^6$ CPU cycles [56]
Input data size (s_j)	1 MB, 10 MB [57], [58], [59]
Arrival rate of requests for task (λ_j)	Poisson distributed with average 10 requests/s [60]

3.7 Results

In the first simulation campaign, performance is assessed when varying the rate of computing task requests for different settings of the maximum delay constraint for the tasks, while in the second simulation campaign, the proposal behaviours have been assessed for various configurations of input data size.
Values are presented with 95% confidence intervals after being averaged across 200 runs.

3.7.1 Impact of delay constraints

We assessed the performance for two different range of maximum delay constraints of the tasks (ideally two different type of QoS that have to be guaranteed) while varying the rate of computing task requests. The amount of data transferred inside the network is depicted in Fig. 3.4(a). Reasonably, the metric rises as the rate of requests rises for all the schemes taken into consideration because more input data must be sent. As stated in Eq. (3.6), the suggested DMEP achieves the targeted objective by reducing the amount of data transferred. Tasks are offloaded in edge nodes that are as close as possible to producers in order to achieve this goal, provided that they do not saturate. For computing tasks with less stringent latency requirements, the minimum data exchange condition is simpler to verify. Instead, when more stringent delay limits are taken into account, the amount of exchanged data traffic is higher. In order to meet them, tasks may be carried out at distant edge nodes, and some of them must also be offloaded to the cloud, as indicated in Fig. 3.4(b). Taking into account the delay

constraint in the optimization problem allows to limit the overall computation delay, which would not have been met otherwise, as illustrated for the MEP benchmark in Fig. 3.4(c). The likelihood that the delay constraint is not satisfied using an oblivious strategy (MEP) is in the range of 0.8 for computing tasks that accept delay constraints of up to 100 ms. When it comes to tasks with less stringent delay constraints, the likelihood is below 0.15. As expected, the Fig. 3.4(c) also shows that the considered metric is equal to zero for DMEP for any delay requirements.

As seen in Fig. 3.5, the lower data traffic transferred in MEP and DMEP as compared to the CloudP benchmark is compensated for by increased latency. Higher computation delays are experienced in Fig. 3.5(c), due to the limited computing capabilities of the edge nodes which incur a longer queuing delay (Fig. 3.5(a)) and execution latency (Fig. 3.5(b)). The queuing delay grows as the number of requests rises. Instead, because more capable edge nodes are chosen as task executors to prevent the saturation of less capable ones, the execution latency actually slightly decreases.

3.7.2 Impact of input data size

In Fig. 3.6, the second campaign results are shown. They encompass two scenarios: the former represents the case in which there is a heavy load with 10% of tasks requiring 1 MB of input data size and 90% of tasks requiring 10 MB of input data size (curves denoted as *J(1MB)=10%J, J(10MB)=90%J*), while the latter represents the case in which there is a lighter load with 10% of tasks requiring 10 MB of data and 90% of tasks requiring 1 MB of input data size (curves labeled as *J(1MB)=90%J, J(10MB)=10%J*). Unsurprisingly, the amount of exchanged data increases as the rate of requested tasks and with the increasing of the percentage of heavy data input size. (see Fig. 3.6(a). Moreover, a increasing percentage of tasks are offloaded to the cloud when the percentage of tasks with smaller input data rises, as seen in Fig. 3.6(b). This is due to the fact that the network is not overloaded by the little amount of data. The amount of input data has no impact on computation time (Fig. 3.6(c)). Despite the allocation of the computing task in the chosen executor may differ to fit the minimization of the objective function, the average computing workload, $\bar{l}$, is the same regardless of the size of the input data that is taken into consideration.

3.8 Open Issues

This chapter has been about DMEP, the proposed strategy for the placement of *CPU-intensive* and *delay-constrained* computing tasks into the edge-cloud continuum.The goal of the optimization problem, which has been formulated as an ILP problem, is to reduce the amount of data exchanged into the edge domain while still meeting the maximum delay constraints of each computing task and not violating the saturation condition of edge nodes, which are modeled as M/M/1 queues. The above requirements are satisfied by not restricting

task execution to the nearest edge nodes, rather enabling farther edge nodes in the edge topology to participate, including the remote cloud. Obtained results illustrate how effective the strategy is when compared to a baseline cloud-only solution. In comparison to the cloud, the amount of data exchanged is drastically decreased (an order of magnitude), but this benefit comes at the cost of a higher computation delay that is controlled within acceptable limits.

Albeit the proposed strategy reaches the objective, it is too complex to be executed, indeed, it is not feasible and scalable because it takes too much time to reach the solution. It can be traced back to a NP-hard problem, called Generalized Assignment Problem (GAP), as you can see in the following lemma.

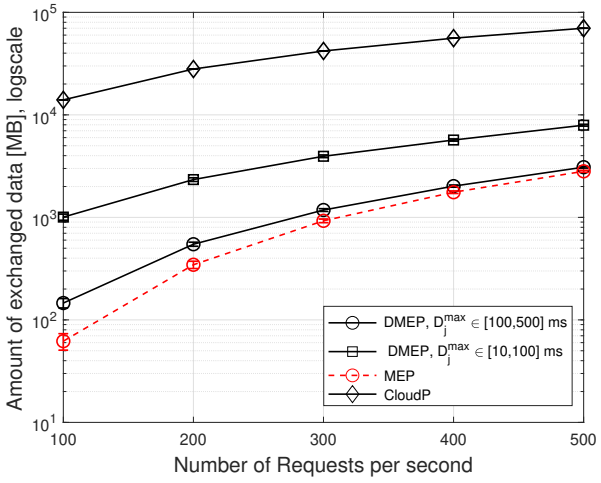
Lemma. *The formulated task placement problem is NP-hard.*

Proof: Our stated problem in Eq. (3.6), is analogous to the GAP, a well-known NP-hard problem in the combinatorial optimization literature [61], [62]. The goal is to assign items to bins (or, in our formulation, compute tasks to edge nodes) as efficiently as possible while reducing overall costs. Each item has a variable size and cost depending on the bin it is assigned to, and bins come in a variety of capacities. The solution is subject to the these constraints: *(i)* each item should be assigned to a single bin (meaning that only one edge node can carry out a task (Eq. (3.7)); *(ii)* neither the capacities of bins nor items' constraints should be exceeded or omitted (i.e., a node's CPU load should not be higher than the level necessary to adhere to the task constraint on computation delay, Eq. (3.8)).

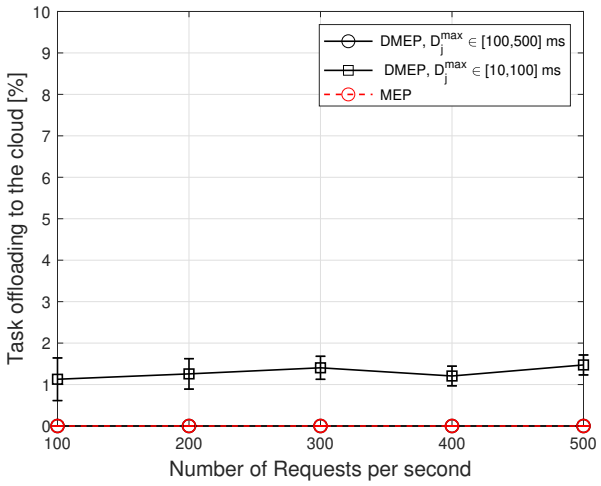
Given the impracticability of the optimal proposed solution, there is the necessity to solve the problem in a feasible way, even if at the cost of using sub-optimal results.

Next chapters will face the issue by designing and evaluating:

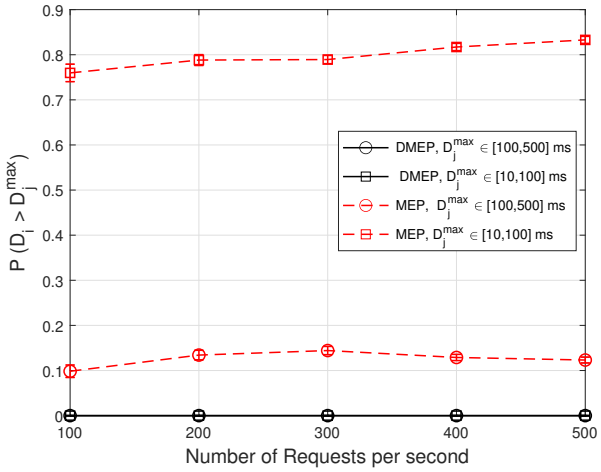
- a series of ML-based solutions (see Chapter 4);
- a heuristic (see Chapter 5).



(a) Exchanged data.

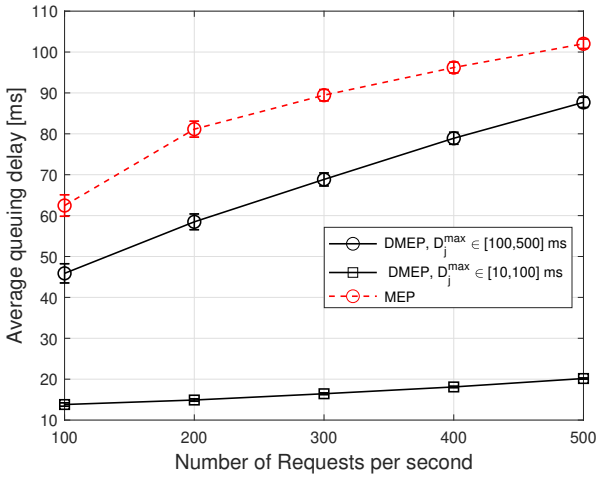


(b) Task offloading to the cloud

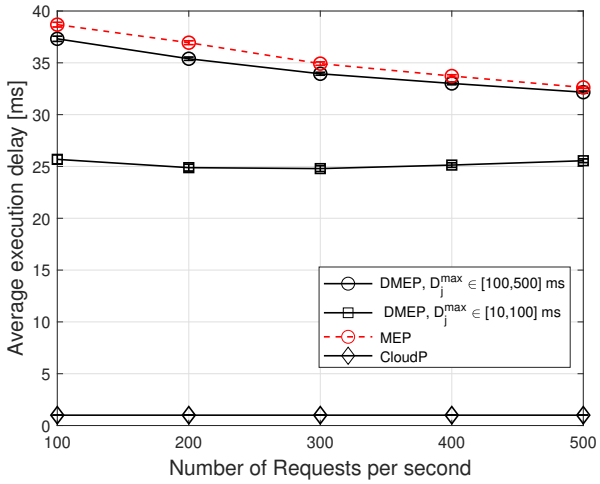


(c) Probability that delay constraints are not met.

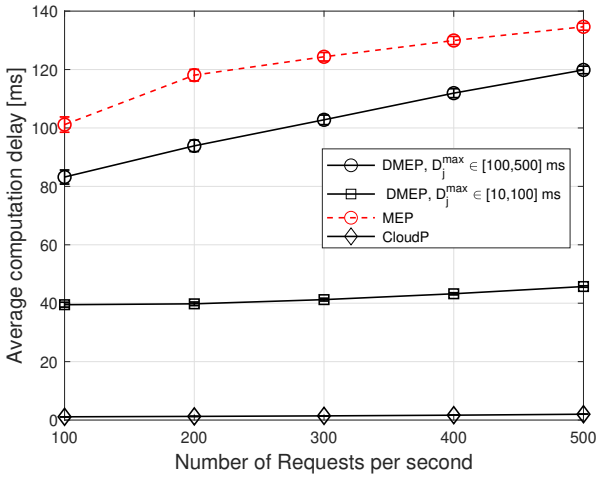
Fig. 3.4. Effectiveness metrics when varying the number of task requests per second ($s_j = 10$ MB).



(a) Average queuing delay.

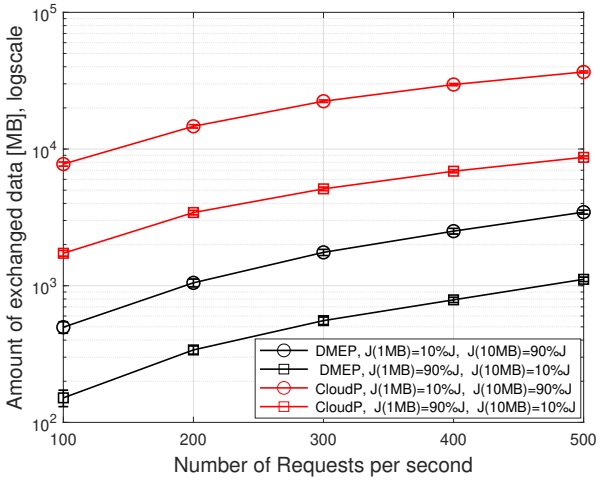


(b) Average execution delay.

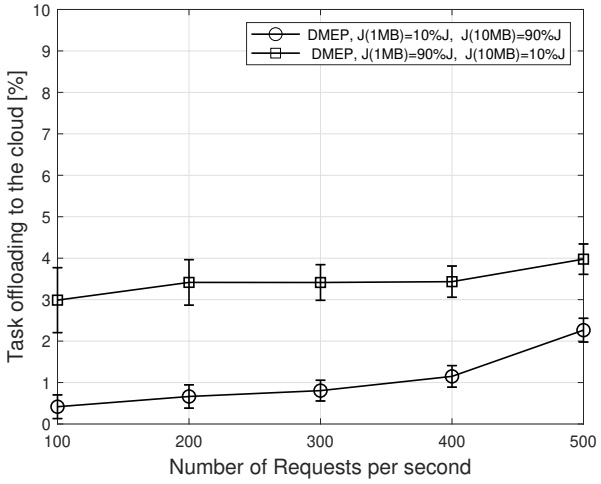


(c) Average computation delay.

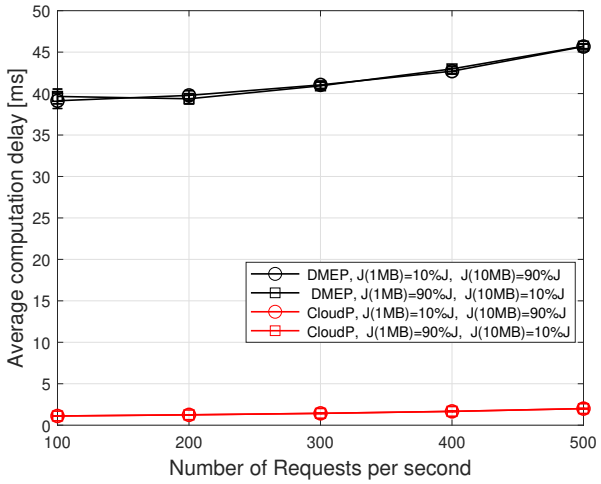
Fig. 3.5. Latency metrics when varying the number of task requests per second (s_j = 10 MB).



(a) Exchanged data.



(b) Task offloading to the cloud.



(c) Average computation delay.

Fig. 3.6. Metrics when varying the size of the input data ($D_j^{max} \in [10, 100]$ ms).

4

Machine learning for task offloading at the edge

Future 6G networks will leverage the complementarity of Edge Computing and Machine Learning (ML) to develop an "intelligent edge", where communication and computing resources will be collaboratively orchestrated. In this chapter, ML algorithms are leveraged to orchestrate the placement of delay-constrained computing tasks in the softwarized edge domain. An overview of the literature about the use of AI in the computation offloading topic are given. The next Sections discuss several and popular supervised learning algorithms, i.e., Decision Tree (DT), Bagged Trees (BTs), Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), that have been implemented to solve the stated problem in a dynamic and efficient manner. Extensive simulation results are given in terms of model accuracy, complexity, and network-related variables to demonstrate how well the methodologies under consideration perform.

4.1 Background and motivations

Some of traditional task allocation strategies rely on the solution of computationally expensive NP-hard problems [63]. This is done regardless of the specifically targeted optimization criteria, such as minimization of energy consumption, network bandwidth, and latency [29], [30]. The already proposed solution (DMEP) in Chapter 3 is not different. As explained in Section 3.8 there is the necessity to find a feasible way to solve the problem.Near-optimal complex heuristics can be envisioned, but they are difficult to implement [64].

Incorporating basic ML techniques in the edge for the dynamic, adaptive, and effective orchestration of communication and computing resources is the goal of the recently proposed idea of the *intelligent edge* [65], which is considered a "crucial 6G enabler" (see section 2.6). ML enables a system's mathematical modeling to be replaced with a data-based management of the system [66]. Intelligent edge could be a possible solution to solve the problem in a different way, compared to a heuristic.

For the purpose of determining if and how to offload computer tasks, supervised learning-based and unsupervised learning-based solutions can be developed, as surveyed in [67]. To

give near-optimal heuristic-like offloading decisions in a scalable and effective way, supervised learning techniques stand out among the others [68].

Most research on intelligent edge solutions currently focuses on task offloading in the presence of either a single edge server [69] or a small number of purpose-built edge servers, such as those co-located with the base stations of a Radio Access Network [67]. There aren't many studies available that thoroughly evaluate the performance of ML algorithms in edge domains with numerous network nodes that have different communication and computation capabilities and could be used as task executors, as it is in this contribution.

ML techniques can be used to make near-optimal heuristic-like decisions, as opposed to creating mathematically tractable heuristics or breaking down complicated issues with high degrees of dimensionality into smaller sub-problems to make them less complex [64], [67]. The work in [65] states that by incorporating edge intelligence into the procedures for allocating tasks and resources, even in extremely dynamic situations where new users are joining constantly, the performance can be optimized.

The work in [70] focuses on the topic of Virtual Network Function Placement and Chaining (VNF-PC), which is the process of allocating network resources to provide Service Function Chains (SFCs). Following the development of an ILP model, the authors create a hybrid technique that combines the best approach with unsupervised ML to locate the necessary network resources while reducing resolution time. The application of the k-means algorithm specifically takes into account the geographic characteristics and request history of the edge servers. Results reveal that the hybrid technique reduces algorithm execution time by up to 75% when compared to the ILP method without sacrificing the provider's revenue or acceptance rate. Nevertheless, the k-means approach is ineffective in highly dynamic situations, because clusters (whose number must be predetermined) may be of different sizes and densities, according to [65].

Meanwhile, scalable, adaptable, and computationally efficient supervised learning methods like DT and SVM deliver near-optimal heuristic-like offloading decisions [68], [71]. For instance, the Classification and Regression Tree (CART), a well-liked DT technique, is taken into account in [72] to choose the appropriate fog devices to which tasks should be offloaded. The maximum computation delay in processing the offloaded tasks is not taken into account in the choice, though. To determine where Virtual Network Functions (VNFs) should be placed, the work in [63] also makes use of a CART algorithm. In [73], classification problems are used to apply an SVM model to the decision of offloading subtasks from cars to edge servers. Additionally, in [74], a task orchestration strategy based on machine learning is put out for the vehicular environment. There, after evaluating various methods, an MLP model was selected as a highly flexible classifier model that forecasts if the results of the offloading decisions are effective or not for each target device.

Convolutional Neural Networks (CNNs) belong to another well-known class of supervised learning algorithms used in edge intelligence contexts [65]. For instance, by taking into account a user tolerance limit, the work in [75] focuses on Automatic Speech Analy-

sis (ASA) tasks to be processed with the least amount of delay in a cloud-edge environment. Despite their excellent accuracy, CNNs still need a lot of training time, which makes them more difficult and expensive in terms of computation than other supervised learning techniques [76], [77]. Additionally, compared to DT and SVM techniques, CNN has inferior explainability because it is mostly based on black box models [78]. The model explainability performance of MLP is improved by its easy hyperparameter adjustment and considerably simpler structure when compared to CNN. Due to this, CNNs will not be used in the study that follows; instead, it will be concentrating on DT, SVM and MLP techniques.

4.2 Proposal

In order to *efficiently* solve the optimal computing task placement problem already stated in the previous chapter (see chapter 3, section 3.5), we concentrate on the same edge domain which involves several heterogeneous nodes acting as candidate task executors, see Fig. 3.3. It is important to emphasize that supervised learning algorithms are trained using the offline optimal solution. As a result, it is possible to make placement decisions that are less complex and speedier at the expense of lengthy training procedures that must be carried out offline.

The specified ILP problem in Eq. (3.6) is equivalent to the Generalized Assignment Problem (GAP), a well-known NP-hard problem in the combinatorial optimization literature [61]. Since it takes a lot of time to determine where the computing tasks should be placed in a realistic environment, it is not possible to face this problem using a standard optimization solver. Therefore, a strategy is required to tackle the problem in a reasonable time, even if it results in a less-than-ideal solution.

We propose to use ML techniques for this purpose, in line with the recent research [67], and, in particular, we chose to trace the task placement problem back to a *multi-class classification problem* since the output can only take discrete values. In particular, all the proposed ML-based techniques are intended to forecast the output, i.e., to choose the task executor node, in response to an offloading task request. The ML-based methods are intended to be run as an application-layer module that interacts with the SDN controller through its Northbound interface. The SDN controller does indeed record various features that can be used as input to the classification models thanks to its view of the edge domain (i.e., network topology, status of links, and available computing resources of nodes) obtained by interacting with overseen nodes through the southbound interface [28]. Then, the SDN controller uses the outputs of ML-based techniques to select the edge node in charge of carrying out the requested offloaded tasks

As we do in our study, the feasibility of imposing ML-based task placement through a centralized SDN controller has also been looked into in the works [79], [80], [81]. We take into account a more challenging case, i.e. a softwarized edge domain with distributed in-network computing capabilities [25], in contrast to the related research.

In the next sections, we specify our multi-class classification problem's input and output more formally, as published in [82].

4.2.1 Input

We associate each task m with an input vector, $\overline{j_m}$, that is characterized by F features $\overline{j_m} = (j_{m,1}, j_{m,2}, ..., j_{m,F})$. After a rigorous feature selection process in which several options were tested, we chose the best configuration (w.r.t. the training accuracy) of number of features $F = 2 \cdot |\tilde{N}| + 1$, where $|\tilde{N}|$ is the number of the potential executors in the edge domain.

Thus, the input vector is made as follows:

- $j_{m,1}$: is the maximum computation delay of the task 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:

$$j_{m,i+1} = \lambda_m \sum_{k_m=1}^{K_m} s_{k_m} \Omega_{i,k_m}^{min} \tag{4.1}$$

- $(j_{m,i+|\tilde{N}|+1})$ with $i = [1, \ldots, |\tilde{N}|]$: is a boolean vector of $|\tilde{N}|$ elements which capture the constraints of the stated problem. Specifically, the i-th position value of the vector will be 1 if the requirements stated by Eqs. (3.8) and (3.4) are satisfied for node i, otherwise it would be 0.

4.2.2 Output

Each ML algorithm outputs a number $y \in \tilde{N}$ that represents the potential executor chosen to carry out the requested task in response to the m-th input. The algorithm, running as application-module of the SDN architecture, chooses the executor. Then, through the SDN controller informs the node of the computing tasks duty, and build the necessary routing ways for the task provisioning.

Misclassification repair

Being ML-based task placement approach, the output may suffer from misclassification errors. When tasks exhibit strong delay restrictions, misclassification difficulties might become very serious. Tasks may produce meaningless results for the users who requested them if they are not completed by the deadline, and their execution may waste resources. In order to correct them, it is proposed also a enhanced version of the algorithms using a post-processing step that would be carried out by the task placement policy that is alerted by SDN controller. The network application in charge of task placement examines if the delay limitations are satisfied for a temporary allocation provided by the deployed ML-based approach for a specific task request. If such is not the case, the application chooses to immediately allocate that task into the cloud to meet the latency limitation, even though doing so would need more data to be transferred into the edge domain.

4.2.3 Dataset generation

The dataset for our experiment was created by taking advantage of the optimal solutions from Section 3.4. In particular, $10,000$ runs were used to create the dataset, which produced $487,997$ in samples (task requests). According to the k-fold cross validation method [83], [84], the created database has been divided into the training and validation sets ($k = 10$). The method uses random sampling to partition the dataset into 10-folds of equal size while excluding repetition. The models are so trained 10 times. A separate 1 fold is utilized as the validation set and the rest folds are used as training sets for each training session. The prediction error is calculated as the mean average of the 10 individual errors made throughout each training session at the end of the process. As a result, using the aforementioned process, it is possible to acquire an evaluation of the observed model's performance that is not influenced by how the dataset was divided into training and validation subsets.

The generated dataset are common to all of the evaluated algorithms, which are described in the following.

4.2.4 ML-based techniques

The formerly defined multi-class classification problem can be solved using a variety of supervised machine learning approaches, as thoroughly survayed in [85]. After evaluating their training performances without optimizing their hyperparameters, has been chosen a subset of them. In particular, the ones which provided a training accuracy more then 40% have been deeply investigated. It is worth to notice that the considered approaches are aligned with literature on task offloading in the field of edge computing [63], [67], [69], [72], [74] to further determine their suitability to address the issue at hand: *(i)* DT, *(ii)* BTs, *(iii)* MLP, *(iv)* SVM.

Table 4.1 lists the main hyperparameters utilized for training each of these strategies, which are explained in the following sections.

Decision Tree

Utilizing a binary tree-structured classifier, the DT approach [86] predicts the outcome in front of an input. The tree's nodes each represent a check to see if the value of a feature in the input vector matches a specific requirement. The algorithm continues going down to the *right* child if the requirement is satisfied; otherwise, it goes down to the *left* child. Each leaf node has a response attached to it. It implies that, given an input, the response will be the class related to that leaf node if a series of choices lead to that leaf node, i.e. the executor node for a given computing task.

Table 4.1. Main hyperparameters and settings for the training of considered ML techniques in MATLAB®.

ML algorithm	**Setting**
Decision Tree	• Decision Tree algorithm: CART [86] • Maximum number of split: 500 • Split criterion: Maximum deviance reduction • Surrogate decision splits: Off • Optimizer: Bayesian Optimization [87] • Acquisition function: Expected improvement per second plus
Bagged Trees	• Ensemble method: Bag [88] • 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 [89] 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 [87] • Acquisition function: Expected improvement per second plus

Bagged Trees (ensemble method)

BTs were created to enhance the performance of a single DT. The fundamental concept is to aggregate the results of each decision tree while taking into account many trees at once. By selecting random and repeatable samples from the original dataset, numerous training sub-sets are produced in order to acquire multiple DTs. Thus, a distinct DT is trained using each subset [90].

Multi Layer Perceptron

The input layer, the output layer, and one or more hidden layers are the three or more layers of neurons that make up the feed-forward Artificial Neural Network (ANN) class known as MLP [91]. Each neuron has a unique nonlinear activation function; classification problem frequently use the *softmax activation function*. When tackling this kind of challenges, like in our study, an MLP is used to categorize an object starting from a specific set of attributes that characterize that object. Each neuron in the output layer represents to one of the potential classes an object could belong to, whilst the number of features to be taken into account corresponds to the number of neurons in the input layer. During the training, the weights between the neurons are tuned so to respond to each input vector (that includes the features of the input object) by activating just the output neuron that corresponds to the predicted class of the object characterized by those features.

Support Vector Machine

Given a set of objects characterized by F features, an SVM [92] 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.
PAR
An SVM [92] classifies a set of objects with F features by locating the *best hyperplane* in the *F-dimensional* features space that divides all the data samples of one class from those of the other class. The hyperplane with the greatest separation between the two classes (or between each couple of classes if it deals with a multi-class classification problem) is known as the *best hyperplane*. The closest samples of two different classes are called Support Vectors. They are used to determine the hyperplane that results from the distance between classes. SVMs can effectively do non-linear classification in addition to linear classification, as previously discussed, by projecting the original features into a higher-dimensional space using a non-linear kernel function. In our study, we also took into account SVM algorithms using quadratic and cubic kernel functions in addition to linear SVM.

Table 4.2. 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 [50]
Task catalog size	1000
Number of hops	• towards an executor in the edge [1,5] hops; [51], [52] • towards the cloud [14] hops.
Maximum computation delay (D_j^{max})	• uniformly distributed in [10, 100] ms [53], [54], [55] • uniformly distributed in [10, 50] and [50, 150] ms [93], [94], [95], [96]
Average task computing workload ($\bar{l}$)	$10 \cdot 10^6$ CPU cycles [56]
Input data size (s_j)	10 MB [57], [58], [59]
Arrival rate of requests for task (λ_j)	Poisson distributed with average 10 requests/s [60]

4.3 Performance evaluation

The objective of the analysis is to compare the performance of the suggested ML-based task placement strategies to the DMEP solution. The goal of each of the compared methods is to select the ideal edge nodes where the tasks should be allocated in order to reduce the transferred intra-domain traffic while meeting the latency restriction.

For ML solution, simulations have been conducted by using the Neural Net Pattern Recognition Tool of MATLAB®.

4.3.1 Settings and metrics

The reference scenario is the same viewed in Section 3.4 for DMEP proposal, while the main settings are reported in Table 4.2. The latter has been updated compared to the Chapter 3 in the maximum computation delay settings and in input data size.

As explained in Section 2.6, each AI proposal integrated with the edge computing paradigm has to guarantee the QoE that is determined by jointly considering multi-criteria, in particular, performance, cost, efficiency and reliability. In alignment with such guidelines, the evaluation of all proposed ML methods is provided in terms of execution times, as well as in terms of traditional ML-related metrics, like accuracy and precision, in terms of meaningful network- and task- related performance metrics to assess their suitability to tackle the problem at hand.

To thoroughly assess the supervised learning algorithms, four main classes of metrics are derived.

Time complexity. Its goal is to measure the efficiency and scalability of the supervised learning approaches under consideration.

ML-related metrics. A set of multi-class metrics, including accuracy, macro average precision, macro average recall, and macro average F1 score [97], [98], are derived after the confusion matrices are extracted from the traditional binary classification metrics [99]. For the sake of clarity, Table 4.3 does not explicitly include micro average precision, micro average recall, and micro average F1 score because they are equivalent to the accuracy measure.

Network usage metric. It is calculated as the total amount of input data (in bytes) traverse from producers to the chosen executors multiplied by the number of network hops that the data traveled through. It is expressed in terms of *data exchanged* in the domain. This indicator illustrates how well the idea works to reduce network utilization.

Task-related metrics. They serve to evaluate how the softwarized intelligent edge acts for the requested tasks, and include:

- *Task offloading to the cloud*: it represents the percentage of requested computing tasks offloaded to the cloud.
- *Task deadline missing probability*: it measures the likelihood that the executed tasks' computation times will be longer than their maximum computation delays. It could be viewed as reliability metric.
- *Average queuing time*: it measures the average queuing time tasks wait in queue before being executed at the chosen edge node.
- *Average execution time*: it represents the average time needed for the accomplishment of the requested tasks.
- *Average computation time*: it is the sum of the average queuing time and the average execution time.

Task offloading to the cloud and *task deadline missing probability* are two of them that demonstrate how well the various approaches do at placing the tasks while meeting their deadlines.

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

4.3.2 Time complexity analysis

Every simulation campaign has been run on an Intel Xeon Gold 6240, with a 2.6 GHz CPU, 18 cores, 32 GB of RAM, and 1 TB of storage (HD). The goal of the first set of results, which are shown in Fig. 4.1, is to compare the time complexity of the various supervised learning algorithms under heavy load status to the DMEP solution. Moreover, the scalability of the methods, whose execution time inexorably rises with the number of requests, is shown in Fig. 4.1.

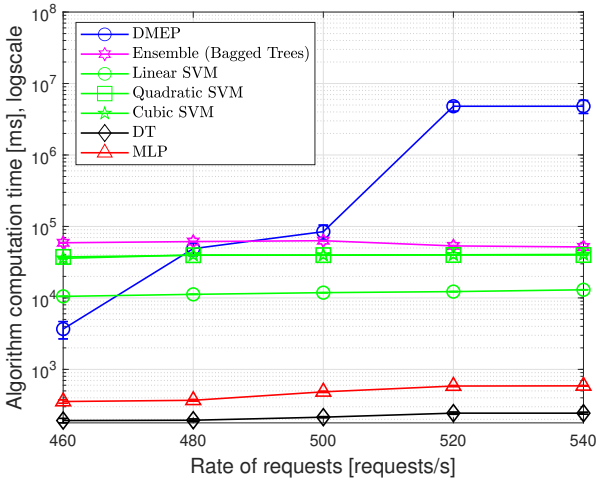


Fig. 4.1. Comparison of the considered supervised learning approaches against the optimal solution (DMEP) in terms of computation time ($D_j^{max} \in [10, 100]$ ms).

Due to the required thorough search to discover the best solution, as was predicted, the execution time of the DMEP solution substantially increases with the rate of requests. DT and MLP are the most effective solutions among supervised learning techniques, that scale well with the amount of requested tasks, being both much below 1s. The algorithm computation times for the other ML-based solutions, which are all above 10s, are too high. Therefore, using them is not a practical option.

4.3.3 ML-related performance

The DT and MLP algorithms are the only taken into account in the following due to their lower temporal complexities compared to the other supervised learning approaches.

The ML-related performance metrics are displayed in Table 4.3. The two compared approaches may be seen to behave almost identically, with DT marginally overcoming MLP.

The accuracy for both models is around 0.6, which is determined by dividing the number of correct task executor predictions by the total number of samples. Strangely, all other indicators are higher than 0.6. The macro average precision for DT and MLP, which is calculated as the arithmetic mean of all the precision values for the various classes (i.e., edge nodes), is specifically 0.66 and 0.62. The arithmetic mean of all recall scores for various classes, which is how the macro average recall is calculated, is 0.63 for both strategies. For both models, the macro average F1 score, which calculates the harmonic mean of the macro average precision and recall, is higher than 0.63.

To better understand the performance results and their impact on task offloading, starting from the confusion matrices, we derived in Tables 4.4 and 4.5 the following probabilities for DT and MLP, respectively: Starting with the confusion matrices, we determined for DT and MLP the probabilities in Tables 4.4 and 4.5 respectively, to better understand the performance outcomes and their effects on task offloading. These probabilities are:

- P_c: is the probability that a specific task will be successfully placed in the appropriate node by the analyzed algorithm (DT/MLP) (i.e., the one foreseen by the optimal solution).
- P_{next}: is the probability that a task, which the optimal solution would have assigned to a specific node, is actually assigned by the analyzed algorithm to a neighboring node in the same layer of the hierarchical topology.
- P_{up}: is the probability that a task, which the best solution would have assigned to a specific node, is actually assigned by the analyzed algorithm to a neighboring node in the immediately upper tier (excluding the cloud) of the hierarchical topology.
- P_{down}: is the probability that a task, which the best solution would have assigned to a specific node, is actually assigned by the method under consideration to an adjacent node in the immediately lower tier of the hierarchical topology.
- P_{others}: is the probability that a task, which the best solution would have allocated to a specific node, is actually placed by the analyzed algorithm in any other node other than those that are adjacent, in the immediately lower and upper layers.
- P_{cloud}: is the probability that a task will actually be placed in the cloud by the analyzed algorithm rather than being allocated to a specific edge node as recommended by the best solution.

Results are computed for every level of the studied hierarchical edge topology, from the remote cloud to the ingress nodes (level 4). It is important to note that tasks that are anticipated to be placed into the cloud, into level 1 nodes, and into level 4 nodes, respectively, do not have probability defined for P_{cloud}, P_{up}, and P_{down}. The values of the P_c columns in Tables 4.4 and 4.5 clearly demonstrate that DT and MLP approaches occasionally may fail to select the same node (the best one) as the optimal solution for the execution of a particular task. However, as shown by the values of the P_{next}, P_{up}, P_{down} columns in Tables 4.4 and 4.5, the executor chosen by ML techniques is likely to be close to the optimal one. The low values of the P_{others} columns in Tables 4.4 and 4.5 demonstrate how rarely the chosen executor is more than one hop away from the best one for each task request. The metrics of concern, such as the quantity of data transferred and computation time, are not greatly impacted by moving the task to a neighboring node since the optimal solution is still adequately approximated (as it would be clearer in the following). This is so because, in our scenario, next nodes are expected to deliver similar computation speeds because they share the same processing resources.

4.3.4 Supervised learning algorithms Vs. DMEP

For the following schemes: DMEP, DT, and MLP, Fig. 4.2 presents the efficacy metrics when the rate of computing task requests is changed. Moreover, curves labeled as MEP are presented, which are indicative of the defined ILP problem without taking the maximum

Table 4.3. 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 4.4. 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

Table 4.5. Topology-aggregated confusion matrix for the MLP ("-" stands for an ineligible probability).

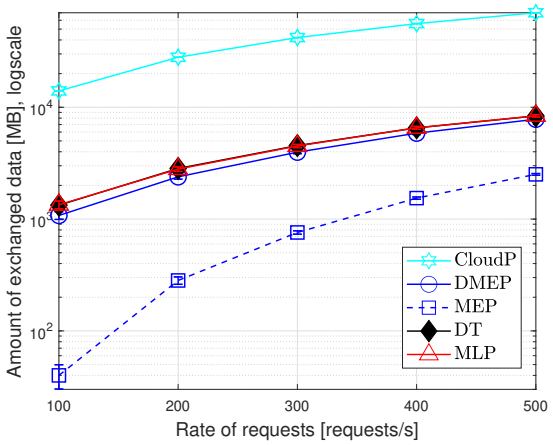
	P_c	P_{next}	P_{up}	P_{down}	P_{others}	P_{cloud}
Cloud	0.962	0	0	0.027	0.011	-
Level 1	0.347	0	-	0.434	0.030	0.189
Level 2	0.550	0.186	0.068	0.150	0.045	0.007
Level 3	0.539	0.089	0.164	0.132	0.076	0
Level 4	0.695	0	0.163	-	0.141	0

tolerated delay for task computation into account, allowing for a better assessment of the influence of the delay limitation expressed in Eq. 3.8.

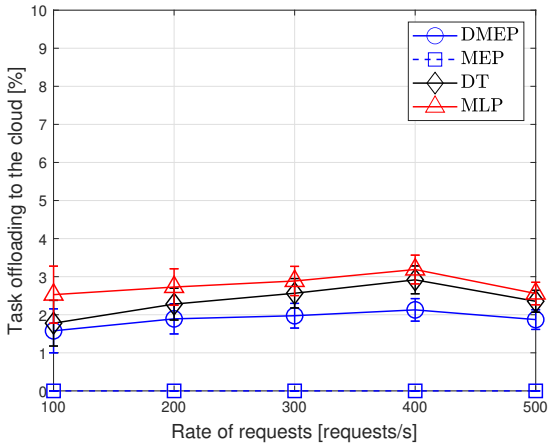
In particular, Fig. 4.2(a) demonstrates that for all the studied schemes, the amount of data exchanged into the edge domain logically grows as the rate of task requests increases since more input data must travel over the network in order to reach the chosen executors.

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

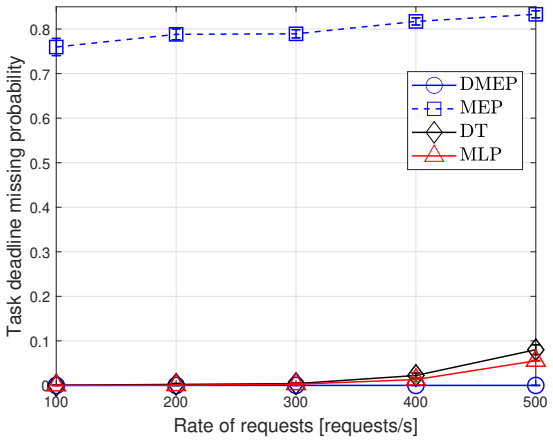
It's interesting to note that the performance of the best DMEP solution are satisfactorily approximated by both DT and MLP. As illustrated in Fig. 4.2(b), all schemes, with the exception of MEP, offload a portion of that tasks to the cloud (up to a maximum of around 3%) in order to meet the computational latency limitations of tasks that cannot be allocated to the edge. This is due to the possibility that, if they are not offloaded remotely, tasks could



(a) Exchanged data.



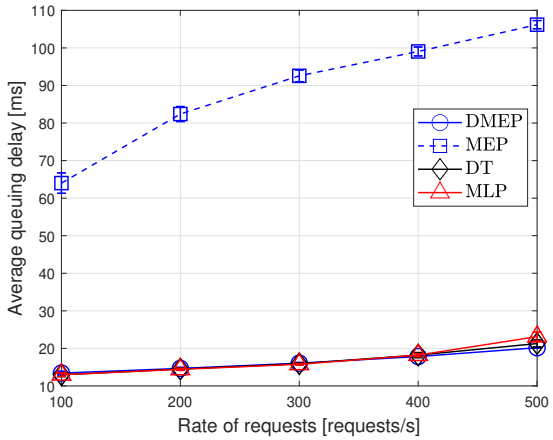
(b) Task offloading to the cloud.



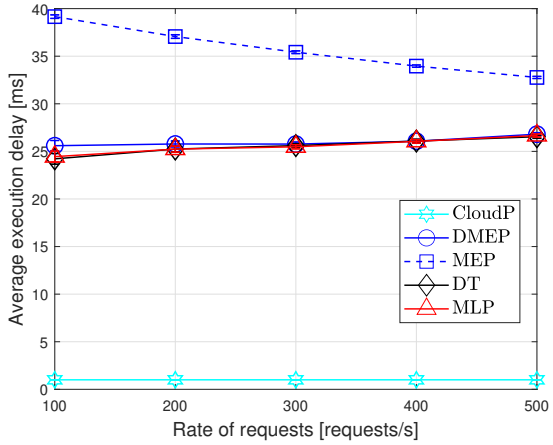
(c) Task deadline missing probability.

Fig. 4.2. Effectiveness metrics ($D_j^{max} \in [10, 100]$ ms).

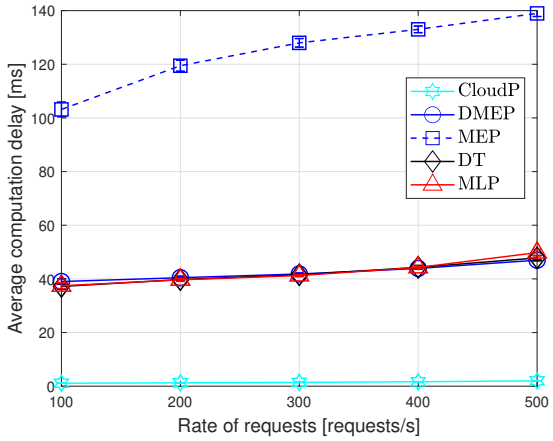
be queued for an excessive amount of time before being processed by edge nodes, which have much lesser computational capabilities than the cloud.



(a) Average queuing time.



(b) Average execution time.



(c) Average computation time.

Fig. 4.3. Delay metrics ($D_j^{max} \in [10, 100]$ ms).

With MLP and DT, meeting such delay restrictions purely depends on the capacity of the used technique to infer the proper placement for each task, in contrast to DMEP, where the delay bounds are provided by hard constraints in the mathematical formulation of the

optimization model (Eq. (3.8)). As a result, even though both DT and MLP attain relatively excellent precision, they occasionally violate the delay bounds because they are unable to forecast the best placement for each task (see Fig. 4.2(c)). Despite these misclassification mistakes, the task deadline missing probability values are, in the worst scenario, on the order of 0.1, which is far lower than the values obtained by MEP, which does not take delay constraints into account when formulating the model. Because no tasks are offloaded to the cloud (Fig. 4.2(b)) as a result, MEP emphasizes task execution at the edge, resulting in the least amount of data transferred (Fig. 4.2(a)). Regarding queueing delay (Fig. 4.3(a)), execution delay (Fig. 4.3(b)), and computation delay (Fig. 4.3(c)), there are no notable differences between DMEP, MLP, and DT. Furthermore, the task request rate has no impact on the metrics.

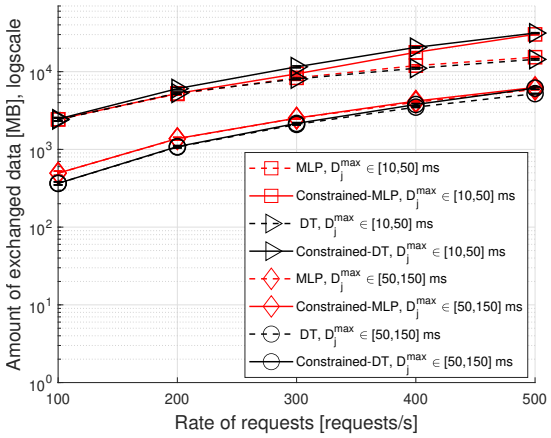
When the rate of requests rises, the average queuing latency in MEP increases more quickly than in the other solutions (Fig. 4.3(a)). Instead, as the rate of task requests changes, the average execution delay reduces (Fig. 4.3(b)). Due to the fact that MEP is indifferent to delay limits, it loads nodes from the weakest to the strongest, which causes task requests to occasionally wait for a very long time before being carried out by edge nodes. These patterns are collectively depicted in Fig.4.3(c) by the calculation delay. Given the cloud's virtually limitless computing capacity, hence the absence of task queuing delays, curves for the CloudP approach are only reported for the execution delay and computation delay metrics (Fig. 4.3(b) and Fig. 4.3(c), respectively). These curves have values that are close to 0.

Additionally, as seen by the limited confidence intervals, findings for all measured metrics reveal trends with no statistically significant variation with an increase in task requests.

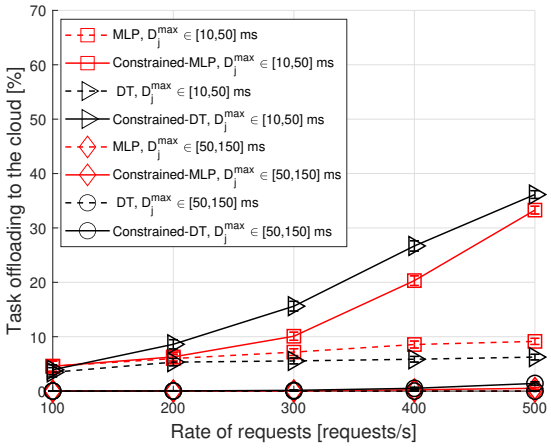
4.3.5 Impact of delay constraints

For the MLP and DT techniques, respectively, the improved schemes explained in 4.2.2 are called *Constrained-MLP* and *Constrained-DT*. The final campaign contrasts the improved schemes with the legacy strategies using various delay settings. In particular, without sacrificing generality, the values of the maximum delay constraints are uniformly distributed in two distinct ranges of $[10, 50]$ ms and $[50, 150]$ ms, with bound values aligned with those frequently taken into account in the literature, [93], [94], [95], to cover a variety of vertical application scenarios, from Industry 4.0 to autonomous driving [96].

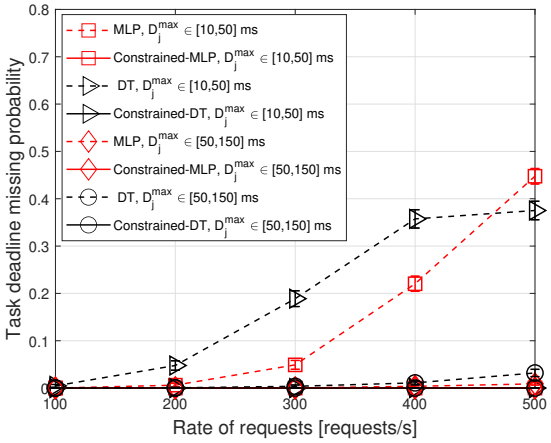
The improved schemes (solid lines) in Figure 4.4(a) display a greater amount of exchanged data than the basic ones (dashed lines). This occurs because, in order to better meet delay restrictions, the enhances schemes offload a bigger proportion of tasks to the cloud that is approximately 35% for MLP and DT with 500 requests/s as illustrated in Fig. 4.4(b). The task deadline missing probability is close to 0, as shown in Figure 4.4(c), in contrast to the legacy MLP and DT techniques, where it approaches 0.45 and 0.4 with 500 requests/s, respectively. The delay values for the comparison schemes are reported in Fig. 4.5. As the



(a) Exchanged data.



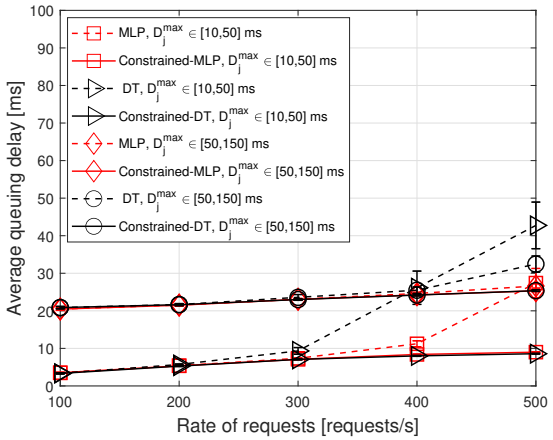
(b) Task offloading to the cloud



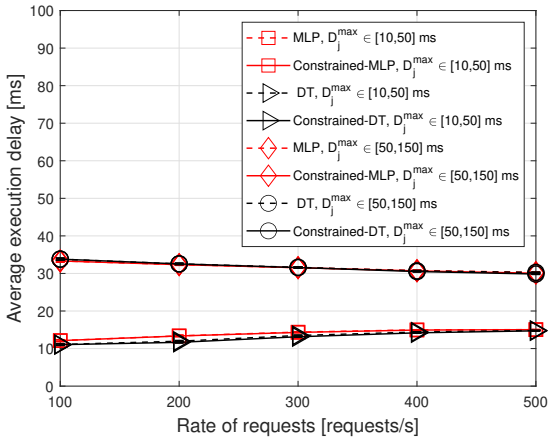
(c) Task deadline missing probability.

Fig. 4.4. MLP Vs. DT: effectiveness metrics.

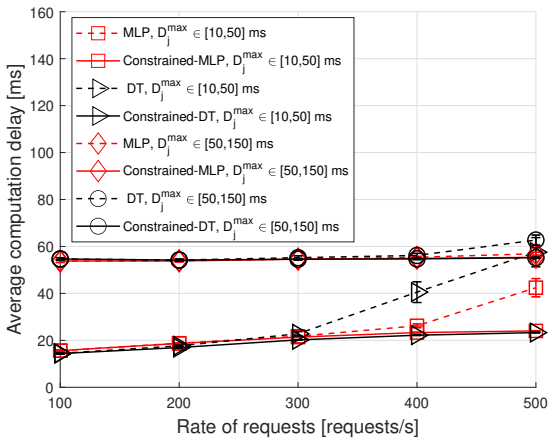
limitations are tightened and the rate of task requests rises, it can be seen that the advantages of the upgraded schemes are becoming more notable (Fig. 4.5(a) and Fig. 4.5(c)). In fact,



(a) Average queuing time.



(b) Average execution time.



(c) Average computation time.

Fig. 4.5. MLP Vs. DT: delay metrics.

the legacy and limited techniques exhibit greater disparities. This is because the legacy methodologies could run into more misclassification concerns as delay limits stiffen.

4.4 Main findings

It is extremely difficult to place computing tasks with stringent performance requirements to heterogeneous edge nodes with constrained capabilities. Task allocation cannot be effectively solved by standard optimization solvers because it would led to unfeasible time of algorithm computation time, and being also an "online" problem. In this chapter, we illustrated the design and implementation of different supervised learning methods that are frequently used in the literature to tackle multi-class classification problems. In particular, we tailored these strategies to address the defined task placement optimization problem in a efficient and feasible manner.

Even though these strategies demand extensive training, the latter ones can be run offline, therefore they have no impact on how quickly a task placement decision is made. Particularly, MLP and DT show to be the most effective methods for the task allocation decision in terms of execution time, with DT reaching slightly lower execution time values than MLP.

Results obtained under a wide range of conditions demonstrate that MLP and DT, in addition to being computationally economical, provide a good approximation of the solution obtained using a standard optimization solver. In fact, they both achieve the defined optimization problem's goal of decreasing the amount of data transferred in the edge domain successfully. When compared to DT, MLP performs somewhat better in terms of meeting task deadlines.

Additionally, improved variations of the aforementioned strategies have been developed to address potential misclassification issues brought on by the inability to adhere to task delay limitations, which become especially crucial for projects with stiffen requirements. The effort was to make the solution reliable in order to guarantee a good QoE for the users, which need to offload their tasks to the network. The task is offloaded to the cloud whenever the task placement network application, through the SDN controller view, discovers that the allocation choice from the ML approach, regardless of the suggested executor, does not meet the task delay constraint. Both techniques perform nearly equally with the proposed improvements.

Next chapter will discuss about a different approach to solve the problem in a feasible way, that is purely heuristic-based.

5

An efficient heuristic for the placement of reusable tasks at the edge with minimum data exchange

In this chapter the design and implementation of the heuristic that solves the task placement problem in a feasible and efficient manner is tackled. As a step further, a new optimization problem is formulated which takes into account those situations in which, given an already processed offloading task, its output can be exploited to serve a new request from different consumers within a certain validity time. By avoiding redundant computations and data exchange, caching such results while they are still valid, has the dual benefit of reducing service provisioning time and enhancing edge resource utilization. A literature overview about this topic is provided. The proposed heuristic solution is validated against the solution achieved through a standard optimization solver and another benchmark heuristic in the literature. Performance is evaluated under different simulation settings.

5.1 Motivations

In addition to all the advantages of offloading user tasks to the edge (see section 3.1), caching the program and/or the output (i.e., the outcome) of the computing task at the edge has recently been acknowledged as an effective way to further reduce the user delay, the energy consumption, and the utilization of edge resources (i.e., bandwidth, processing) [29], [100], [101]. While most of the referenced works have concentrated on caching the program, which is frequently reusable [101], [102], caching the computation output has received less attention. This technique, known as *compute reuse* can significantly decrease the amount of redundant data transfers and edge computation and may lead to up to 50x faster task computation times than the case with no reuse, as analyzed in [103].

Several applications can profit from edge compute reuse, whether it be partial or complete. Applications that employ AR can make use (partially or complete) of the same computation result for users who are visiting the same site or attending the same event (such as a concert). The same computational task (for example, the history of that picture) might be requested by many visitors in a museum who are supplying images of the same painting from various angles or with various tones [104]. For reference, the Louvre museum, which has more than 380K items and exhibits 35K works of art, draws about 15K visitors daily.

The countermoves already calculated in front of the identical configurations and past moves of the players can be reused in mobile gaming applications like a chess game, given the limited set of First-Move Configuration (FMC) [29], [103]. This spurs the possibility of massive compute reuse [105].

The term *temporal validity* refers to the fact that results are sometimes only helpful for a little period of time before being removed from the cache. For instance, event-based processing [106] can be extremely popular in that it can deliver results that are highly requested (and thus, reusable) by various consumers, but only for a limited amount of time before the output of the computation loses significance due to changes in the spatio-temporal conditions in which input data are retrieved. Only when approaching the intersection at extremely close intervals of time can many cars at an intersection request the obstacles detected nearby and be served by the same executed object detection task. The tasks whose output is reusable for a specific period of time will be referred to as *time-limited reusable computing tasks* in the following.

Besides solving the issues of our DMEP solution (see section 3.8) , the main goal of this chapter is to devise a novel placement strategy of *time-limited reusable computing tasks* in a network edge domain, aimed *at minimizing the edge resources usage*, by both reducing the amount of intra-domain traffic and better exploiting the edge nodes' computing resources.

5.2 State-of-the-art

The current section is about the state of the art of concept of reusable computing task and of the SDN-enabled orchestration of them.

5.2.1 Reusable computing tasks

Related edge computing research frequently makes the assumption that computing tasks requested by various clients are unique from one another and must be carried out independently [107]. However, the academic community has begun to think about the idea of caching, and thus reuse, the program, the input data, and/or the compute output in order to increase performance.

The advantages of program caching, also known as service caching [102], [108], [109], are well acknowledged in recent literature [101], [102]. Additionally, some research have thought about the potential for caching both the program and the input data, for example, to lower the latency of services and the energy usage of mobile devices [110].

However, there are numerous instances where calculation results can be used again in whole or in part. It is referred to as *content caching* in [101] and *compute reuse* in [104] and [111] when computing tasks outputs are cached in order to reduce unnecessary calculations. In the case of the mobile online game [112], for example, a processed gaming scene may be requested nearly synchronously by individual participants. Certainly, caching and reuse

of the result would lessen the computational effort on the edge nodes as well as the players' experience of service provisioning latency. Additionally, different clients in close proximity may issue requests exhibiting spatio-temporal locality. As an example, in AR applications, visitors or passengers in the same area of a museum or airport may request, nearly simultaneously, the same processed AR output or part of it [29]. In these circumstances, compute reuse is practical due to input similarity, which results from the same contextual information - such as landmarks and road signs - being collected [113].

Reusing computed outputs could be especially beneficial for edge nodes with limited capabilities, since it could prevent the redundant execution of tasks that require a lot of processing power, such as Deep Neural Network (DNN) inference [66]. The work in [112] in this context takes into account a multi-user single-edge server scenario and proposes a time-constrained energy-efficient task offloading strategy with compute reuse. The work in [103], on the other hand, gives an empirical strategy to justify the necessity of compute reuse across consumers and stakeholders. The authors assess the benefit of edge computing systems with compute reuse compared to those that do not have this function by implementing three popular edge computing applications, including matrix multiplication, face detection, and chess. They discover that the reuse greatly reduces CPU utilization and task completion times.

From a networking viewpoint, a major obstacle to compute reuse in the presence of a distributed edge computing infrastructure is to identify different requests for the same task and transmit them to the same edge node that cached the result. This also suggests the definition of specialized methods that [27] uniquely name and define the computing tasks. Information Centric Networking (ICN) paradigms, which implement routing-by-name procedures and explicitly name contents and computations at the network layer, can address these issues, as acknowledged in [100]. The authors introduce ICedge in [104], a first network-based framework that makes use of ICN to provide name-based compute reuse abstractions in a fully distributed fashion. The best task orchestration is not taken into account; instead, the emphasis is on the creation of standard naming standards and name-based forwarding rules.

The output of the reusable computing task might have a *limited time validity*, which can range from hundreds of milliseconds to minutes or even days before it is rendered worthless. This observation should not limit the scope of the discussion. This is a typical feature for activities that accept time-varying data as input, such as environmental conditions or context-aware data gathered from a tourist using the Google Lens app while exploring a city or a security camera in a building. A multi-player mobile gaming application can require various users to see the same scenario that the server has produced. To serve requests occurring close together in time, the output can be calculated once and stored. In interactive games, the maximum allowable latency for a player can range from 100 milliseconds to one second. After that time has passed, the output is no longer useful [114]. The maximum number of players that may be hosted on the same server and the number of players who can reuse the

same output vary by game genre, ranging from tens to hundreds to thousands for massively multiplayer games [115].

In the event of a chess match, replies to a previously computed FMC can be employed for several players over a much longer period of time [103]. Similar to this, a Google Lens app's request for information on a well-known landmark's history may, over time, satisfy a number of users [113].

For the same sort of task, the time validity might change with time. For instance, during peak hours, the output of a task linked to traffic safety has a limited time validity, yet during off-peak hours, when the environment changes more slowly, the time validity may be greater.

5.2.2 SDN-enabled reusable computing task orchestration

The proposed works in section 3.3.2 differ from the proposal in this chapter in terms of the desired optimization objectives, but they both use SDN technology to centrally decide which tasks should be sent to edge nodes. They do not take into account the compute reuse, in contrast to the work in this chapter.

Theoretically, interacting with the SDN controller thorugh the Northbound interface, the proposed solution might direct queries toward edge nodes that can reuse the output of previously completed tasks, as was suggested in [27].

As far as we know, the work in [28] represents a preliminary effort in taking the in-network caching into account in a software-defined edge architecture. There the combination of the Information Centric Networking (ICN) and SDN concepts is used to promote compute reuse . In order to ensure the overall faster service provisioning time, SDN is in charge of controlling routingwhile ICN's native in-network caching capabilities is employed to easily cache calculation results. However, the tasks' limited time validity is not taken into account.

5.3 Proposal

To the best of our knowledge the compute reuse topic is in an infancy stage of research and its impact on the achieved performance has not yet been investigated. Thus, the work of this chapter is motivated by the need to properly account for the time validity of the computation task's output, referred to the edge placement of reusable tasks problem.

We abstract from the specific networking solution of edge nodes that could be IP-based or ICN-based. Instead, starting from the system model illustrated in Chapter 3, we concentrate on the definition of a new optimal strategy for the in-network placement of time-limited reusable computing tasks in a distributed SDN-based edge infrastructure, with the goal of reducing network resource usage while satisfying end-user QoS. In addition, it is worth to underline that, both the optimization problem without considering the reuse and the optimization problem considering the reuse, are solved at first with standard optimization solvers and secondarly with our proposed heuristics. The latter are near-optimal solutions

Table 5.1. Summary of the main notations.

Symbol	Description
N	set of edge nodes
d	cloud node
$\widetilde{N}$	set of candidate executors $N \cup d$
i	generic edge node
μ_i	CPU capabilities of node i (CPU cycles/s)
S_i	caching capabilities of node i
M	set of computing tasks
j	generic computing task
D_j^{max}	maximum delay for task j to be computed
T_j^{max}	time validity of the output of task j
K_j	number of input contents for computing task j
s_j	size of the overall input data for computing task j
s_{k_j}	size of each input content c_{k_j} required for computing task j
l_j	CPU requirement for task j
X_{ij}	binary decision variable: 1 if task j is executed by node i, 0 otherwise
λ_j	arrival rate of requests for task j
λ_j^{exec}	actual execution rate of requests for task j
T_j^{exec}	average time interval between two successive executions of task j
Ω_{i,k_j}^{min}	number of hops separating the potential executor i and the closest providers for each input content c_{k_j} required for task j
p_j^v	probability that the output of a computing task j is still available and valid in the edge domain

and are proved to be feasible and scalable as the reader may see in the performance evaluation section of this chapter. In the following sections, we will outline the upgraded system model and the problem formulation that include the computing reuse.

5.4 System model

The reference scenario (see 3.3) and the assumptions (see section 3.4.1) are the same of that in Chapter 3. In the upgraded system model, new assumptions that are referred to the *compute reuse* aspect of the problem are illustrated. For ease of reference, the key notations used in the thesis are upgraded and summarized in Table 5.1. The computing task, j is included in M, which is the set of tasks ($j \in M$). It can be characterized by the following tuple: $\{s_j, D_j^{max}, l_j, T_j^{max}\}$. Here, s_j is the size of the input data for computation; D_j^{max} is the maximum delay constraint for task j computation; l_j is the amount of computing resources, in terms of CPU cycles, needed to complete task j; and T_j^{max} is the time validity

of j-th task's output (i.e., the output of the computing task j is removed from the cache when T_j^{max} expires because it is deemed worthless and unable to satisfy any more requests).

T_j^{max} pertains to a specific instance of a computing task and is not related to the kind of computing task since for a given type of computing work, the time validity may change depending on the spatio-temporal context. Without losing generality, T_j^{max} in our approach specifically represents the validity of the input data used to fuel the computing task, which is determined by the data providers themselves. For instance, in a smart building scenario, various consumers, such as user apps, Heating, Ventilation and Air Conditioning (HAVC) systems, energy management systems, etc., may analyze and utilise the environmental data that are periodically gathered by the sensors. Environmental variables, such as temperature and humidity, often have a predetermined time validity, such as a few minutes. As a result, the output of a processing task that uses those parameters as input will be valid for the same period of time. Then, additional instances of the same data will be generated, thus a new computation is needed. If the task requires data with varying validity times as input, the executor will choose the lowest value as the task output's reference validity time.

The added new assumptions holding the work are the following[1]:

- In order to process a computing task data must be provided as input to a computer program[2] that, once run, produces an output.
- After a computing task has been completed, it is presumed that the output has a finite shelf life. Requests, which arrive after the time validity has passed, need that the task be run again using fresh input data.
- Computing tasks can *fulfill requests from different consumer* since they are *fully reusable* during their period of validity. The popularity of a task is determined by the arrival task request rate, which only influences how many times the computation output is reused throughout the time validity of the task and not the time validity itself.

5.4.1 Modelling of the time validity of the output

We assumed in section 3.4 that the arrival rate of requests for a computing task j in the edge domain follows a Poisson distribution with parameter λ_j. When the requests are not synchronized and independent of one another, as is the case for the scenario under consideration, the poisson distribution represents a plausible approximation. It is important to note that the proposed framework is still valid as long as the appropriate mean delays are taken into account for a varied arrival rate (e.g., generic).

When an edge node is chosen to carry out a computational task j, it keeps the results of the computation for as long as the time validity T_j^{max} is still valid. There is no need to

[1] See Section 3.4.1 for the old assumptions that are still valid within this chapter.

[2] Since the focus of our research is compute reuse, we assume that each edge node has access to the programs required to complete the task [116]. They may be proactively retrieved from a storage service available in the edge domain.

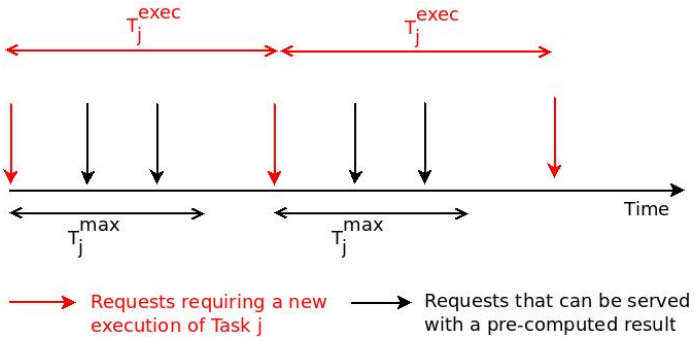


Fig. 5.1. Time period, T_j^{exec}, between successive executions of task j with validity time T_j^{max}.

carry out the same task again (and to retrieve new input data for it) if a request for it comes in within the validity time. Instead, if a consumer makes a request for the same task after T_j^{max}, the task must be computed again and fresh input data must be retrieved from the source(s). The *validity probability* of task j is also known as the "probability that a request may be satisfied by a pre-computed task j" and is represented by p_j^v.

As a result, if j-th task's request arrival rate is λ_j, that task will actually be executed at the rate λ_j^{exec}, which is determined by the equation below.

$$\lambda_j^{exec} = (1 - p_j^v) \cdot \lambda_j. \tag{5.1}$$

As shown visually in Fig. 5.1, we define $T_j^{exec} = \frac{1}{\lambda_j^{exec}}$ as the average time interval between two successive executions of task j.

According to the current hypothesis, T_j^{exec} is greater than T_j^{max}; as a result, similarly to [117], the validity probability p_j^v can be represented as the ratio between the time validity of j-th task's output over T_j^{exec}:

$$p_j^v = \frac{T_j^{max}}{T_j^{exec}} = T_j^{max} \cdot \lambda_j^{exec}. \tag{5.2}$$

By combining Eq. (5.1) and Eq. (5.2), the probability p_j^v can be derived as follows:

$$p_j^v = \frac{T_j^{max} \cdot \lambda_j}{1 + T_j^{max} \cdot \lambda_j}. \tag{5.3}$$

5.4.2 Computation Delay

Consequently, the new computation delay model is updated, thus the task arrival rate at node i (see Eq. (3.1)), the average computation delay at node i (see Eq. (3.3)), and the stability constraints of the M/M/1 queues (see Eq. (3.4)) change to the following equations:

$$\sum_{j \in M} X_{ij} \lambda_j^{exec}, \quad \forall i \in N \cup d. \tag{5.4}$$

$$\overline{D_i} = \frac{1}{\frac{\mu_i}{\overline{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec}}, \quad \forall i \in N \cup d. \tag{5.5}$$

$$\frac{\mu_i}{\overline{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec} > 0, \quad \forall i \in N \cup d.; \tag{5.6}$$

5.5 Problem formulation

Given the new system model, the aims become *(i)* to minimize the network resource usage within the edge domain, by also *exploiting the reuse of computation results*, thus improving the efficiency of the edge nodes in-network computing resources, and *(ii)* to meet the task delay requirements, in order to satisfy the QoS of the end-users. It is worth to underline, that inefficiencies can be experienced if the placement decision is taken while being oblivious of the chance of reusing computing tasks. Hence, the formulated optimization problem specifically accounts for the time validity of the output of a computing task. Through the parameter T_j^{max}, which has an impact on the validity probability p_j^v, the objective function specifically takes into consideration heterogeneous tasks in terms of the time validity of the output. λ_j^{exec}, as expressed in Eq. 5.1, depends on the validity probability of task j, and therefore, indirectly captures the limited time validity of a cached output. It is worth to notice at the outset that the problem formulated in the previous chapters can be regarded as a special case of the problem formulated in this section. Indeed, if T_j^{max} is equal to zero (i.e. it is not possible the reuse of the tasks' output because they instantly expire), p_j^v become zero, so λ_j^{exec} become λ_j .

Thus, the new optimization problem can be formulated as follows:

$$\min \sum_{i \in \widetilde{N}} \sum_{j \in M} X_{ij} \lambda_j^{exec} \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min} \tag{5.7}$$

s.t.

$$\sum_{i \in \widetilde{N}} X_{ij} = 1, \quad \forall j \in M; \tag{5.8}$$

$$X_{ij} \overline{D_i} \leq X_{ij} D_j^{max}, \quad \forall i \in \widetilde{N};\ \forall j \in M; \tag{5.9}$$

$$X_{ij} \in \{0,1\}, \quad \forall i \in \widetilde{N},\ \forall j \in M. \tag{5.10}$$

We can rewrite Eq. (5.7) by expressing λ_j^{exec} through Eq. (5.1) to more clearly show all the variables influencing the current problem, thus obtaining:

$$\min \sum_{i \in \widetilde{N}} \sum_{j \in M} X_{ij} (1 - p_j^v) \cdot \lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}. \tag{5.11}$$

Moreover, the same considerations made in Section (3.5) regarding the constraint on maximum computation delay also apply in this new case. Hence, Eq. (5.9) can be substituded with:

$$X_{ij} \leq D_j^{max} \left(\frac{\mu_i}{\overline{l}} - \sum_{j \in M} X_{ij} \lambda_j^{exec} \right), \forall i \in \widetilde{N}; \forall j \in M. \tag{5.12}$$

5.6 The Conceived Heuristics

The ILP defined in Eqs. (5.7)-(5.10) can achieve the best solution for the task placement problem, but as the problems grows, it becomes more computationally expensive.

We implemented a heuristic approach to increase the speed of the task placement solution search. The goal of the heuristic is to place each computing task, denoted as M_r (with $M_r \subseteq M$), into the edge-cloud continuum nodes that guarantee the lowest cost (see Eq. (5.15)) expressed as the amount of data exchanged to carry out the processing tasks, in alignment with the objective function in Eq. (5.7), while satisfying the delay constraint in Eq. (5.12)

The amount of data sent to execute task j on node i can be measured as follows:

$$Data(i,j) = \lambda_j(1 - p_j^v) \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}. \tag{5.13}$$

The number of operations (measured in CPU cycles) needed to execute a task j, $WL(j)$, can also be calculated as follows:

$$WL(j) = (1 - p_j^v) \cdot l_j. \tag{5.14}$$

The amount of data exchanged for each CPU unit of node i assigned to task j is what we refer to as the cost of executing the task j on node i, $c(i,j)$ (normalization is taken into account to account for heterogeneous task demands and is perfectly in line with the common practice adopted to design heuristics for allocation problems; [62]).

$$c(i,j) = \frac{Data(i,j)}{WL(j)} = \frac{\lambda_j \sum_{k_j=1}^{K_j} s_{k_j} \Omega_{i,k_j}^{min}}{l_j}. \tag{5.15}$$

5.6.1 The heuristic algorithm

The proposed heuristic uses a greedy strategy, which is described below in Algorithm 1, to approximation the optimal task placement solution [3].

The algorithm runs iteratively. The sets of requested tasks and potential executors, that are sorted in ascending order w.r.t. their processing capabilities (line 1), are first taken into account. It calculates the cost of allocating each work $j \in M_r$ to each executor $i \in \widetilde{N}$ according to Eq. (5.15). The executor and cost pairs are added in ascending order (according to $c(i,j)$) for each $j \in M_r$ into a data structure termed the *list of best executors for* j, BE_j (lines 2-8). Basically, the best executor and the associated cost are identified in the first element of the list, followed by the second best executor and the associated cost, etc.

The algorithm chooses a node i at each iteration and attempts to assign tasks to it where i has been recognized as the best executor (lines 11-32). The method, in particular, generates a list T_i (lines 14) of potential tasks to be placed in node i and arranges the items

[3] Notice that in the case the reuse is not enabled, the algorithm is the same because only $DATA(i,j)$ and $WL(j)$ change.

in descending order depending on the cost $c(i, j)$. (line 17). The algorithm then determines if the selected tasks in T_i can actually be allocated in node i depending on whether the contraint stated in Eq. (5.12) is verified (line 21). If the outcome is successful, the tasks are removed from the set M_r; otherwise, alternative placement alternatives must be taken into account based on the data in BE_j. For the sake of clarity, when the outcome is negative the node i is purged from the BE_j, thus the new first element of BE_j is the j-th task's second best executor, which will be taken into consideration in the next iterations. The cycle repeats until every task is assigned to a edge node or in the cloud.

Algorithm 1: The proposed heuristic algorithm

input : Set of candidate executors $\widetilde{N}$; set of requested computing task $j \in M_r$; processing capabilities $\mu_i \ \forall i \in \widetilde{N}$;

output: $X_{ij} \ \forall i \in \widetilde{N} \ , \forall j \in M_r$

sort($\widetilde{N}$, according to μ_i, increasing);
for $j \in M_r$ **do**
  for $i \in \widetilde{N}$ **do**
    calculate $c(i,j)$;
    insert $[i, c(i,j)]$ in BE_j;
  end
  sort(BE_j, according to $c(i,j)$, increasing);
end
while $M_r \neq \emptyset$ **do**
  $i \leftarrow 1$;
  while $i <= |\widetilde{N}|$ **do**
    for $j \in M_r$ **do**
      if *the first element of* BE_j *contains* i **then**
        insert $[j, c(i,j)]$ in T_i
      end
    end
    sort(T_i, according to $c(i,j)$, decreasing);
    while $T_i \neq \emptyset$ **do**
      $j \leftarrow$ ***read(T_i, first element)***
      update $\overline{D_i}$ to include j's load;
      if $\overline{D_i} < D_j^{max}$ **then**
        $X_{ij} \leftarrow 1$;
        purge j from M_r;
      end
      else
        purge i from BE_j;
        restore $\overline{D_i}$ by excluding j's load;
      end
      purge j from T_i;
    end
    i++;
  end
end
return $X_{ij} \ \forall i \in \widetilde{N} \ , \forall j \in M_r$

5.6.2 Heuristics complexity and approximation bound

The complexity of our algorithm rises linearly with the number of nodes and is bounded by $O(|M_r| \cdot |\widetilde{N}| \cdot \log |M_r|)$ if we take an effective sorting algorithm like QuickSort into consideration.

Furthermore, it is possible to demonstrate an approximation bound for the developed heuristic. The maximization of a submodular function under a knapsack constraint is equal to the described problem. We can show that the proposed heuristic, according to [118], can nearly achieve $(1-\frac{1}{e})$-approximation, which is the best result credited to Sviridenko in [119]. Similar to [120], before proving that the approximation holds, we first do a transformation by introducing a new reward function (to be maximized) for each task, i.e., $f_r(i,j) = c(d,j) - c(i,j)$, where $c(d,j)$ is a constant that denotes the expected cost of a given task when it is allocated in in the cloud (denoted as d).

The objective function, $f_r(i,j)$, may be shown to be a monotone, non-decreasing submodular function. Monotonicity is trivial to demonstrate because any new placement of a task cannot make the objective function less valuable. In fact, the following scenarios are possible:

- There will be no additional gain in the case a new task has already been completed and the relevant output is still valid.
- If a new task is computed an edge node, the additional gain is positive (the task requires data transmission in order to be carried out because the input must get to the executor, but this cost is less than $c(d,j)$).
- If a new task is computed in the cloud, the additional gain would be zero.

In terms of submodularity, it is sufficient to demonstrate that the objective function $f_r(i,j)$ is submodular $\forall j \in M$. This is because the sum of submodular functions is still a submodular function. The idea of diminishing returns is captured by submodular functions: as the set gets bigger, the advantage of adding a new piece to the set is smaller [121]. This seems to apply to our problem because as the number of tasks grows, new tasks are more likely to be executed to farther edge nodes (the closer ones become saturated) and ultimately to the cloud, where the gain will not increase at all because input data must travel through more hops (the corresponding cost $c(i,j)$ rises).

5.6.3 Implementation aspects

In a practical design, the Controller uses its broad perspective of the edge domain to carry out the heuristic algorithm. Like in conventional SDN deployments, the Controller keeps a Network Information Base (NIB), which contains all the details about the topology of the edge network as well as the capabilities of the nodes. As a result, the Controller may determine how many hops there are between a generic edge node i and another end-point.

Assumedly, the ingress nodes receive messages containing task requests from various consumers, deliver them to the Controller, and then watch for the configuration of the flow tables. The attributes s_j, D_j^{max}, l_j, T_j^{max} are included in a request message for a task $j \in M_r$. By gaining access to this data and the NIB content, the Controller uses Eq. 5.15 to determine the corresponding cost of performing each task $j \in M_r$ in each edge node $i \in \widetilde{N}$, and execute the Algorithm 1. The Controller fills the flow tables based on the $X_{i,j}$ output to create the routing path from the ingress nodes to the executors and, in parallel, from the executors to the input data providers. Following receipt of the task request, the executors gather the input information from the producers, carry out the computation, and then deliver the results to the consumers.

5.7 Performance evaluation

The objective of the conducted evaluation study is to evaluate the performances of the proposed optimal task placement strategy. We validate the heuristic in a first campaign by contrasting it with the outcomes of the ILP problem that was addressed (DMEP). Matlab® is used to develop heuristic solutions. To evaluate the performance of the proposal in comparison to benchmark placement solutions, a second simulation campaign has been run.

5.7.1 Simulation settings and tools

Network topology. The reference scenario is the same viewed in Section 3.4 for DMEP without reuse proposal, while the main settings are reported in Table 5.2. The latter has been updated compared to the Chapter 3. In the following, the main upgrades.

Task settings. We still consider a catalog of 1000 computing tasks, but in this case they differ not only in computation load and maximum computation delay, but also in the time validity of their outputs. Moreover, unless otherwise specified in the text, we model the popularity of each work j in accordance with Zipf's law, similar to [122].

Metrics. The metrics are the same of those in Section 3.6.3. Only the *Edge computation hit* metric is added in order to evaluate the benefits introduced by the *computation reuse*. It calculates the fraction of task outputs computed by edge nodes that are used to fulfill further requests over the total number of tasks requested.

Benchmark. We implemented a further heuristic that is presented in [123] and we used it as a benchmark solution. The referenced paper, described a set of heuristic to solve the GAP, that are obtained by the local-ratio technique to every algorithm which solve the single 0-1 knapsack problem Moreover, it theoretically demonstrate the optimality bound of the problem.

It is specifically shown how any polynomial time α-approximation algorithm for the knapsack problem may be converted into a polynomial time $(1 + \alpha)$-approximation algorithm for GAP. According to fairly recent literature [124,125], the approximation algorithm in [123]

Table 5.2. 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 [50]
Task catalog size	1000
Number of hops	• towards an executor in the edge [1,5] hops; [51], [52] • towards the cloud [14] hops.
Maximum computation delay (D_j^{max})	Uniformly distributed in [10, 100] ms [53], [54], [55]
Time validity of the output of a task (T_j^{max})	Exponentially distributed with average $\overline{T_j}^{max}$= 300, 500, 1000 ms
Average task computing workload ($\bar{l}$)	$10 \cdot 10^6$ CPU cycles [56]
Input data size (s_j)	1 MB, 10 MB [57], [58], [59]
Arrival rate of requests for task (λ_j)	Poisson distributed with average 10 requests/s [60]
Task requests distribution	• Uniform • Zipf (α=0.8, 1.2) [122]

is among the most well-established ones for solving the GAP, also in the realm of edge computing. In our example, the heuristic takes the cost function (Eq. (5.15)), the node capabilities, and the task constraints as inputs. After reducing the GAP by iteratively solving the 0-1 knapsack problem for each bin, through the conventional greedy algorithm cited in [123], the heuristic returns the set of tasks assigned to each node. Also in this case, simulations results are averaged over 200 runs and reported with 95% confidence intervals.

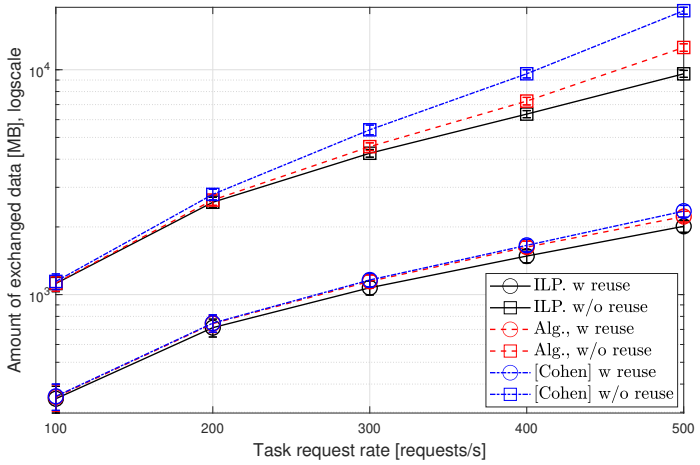
5.7.2 Results

In this section, the heuristics are first validated. Then they are compared with different benchmark solutions, evaluating the impact of tasks' popularity and the output time validity.

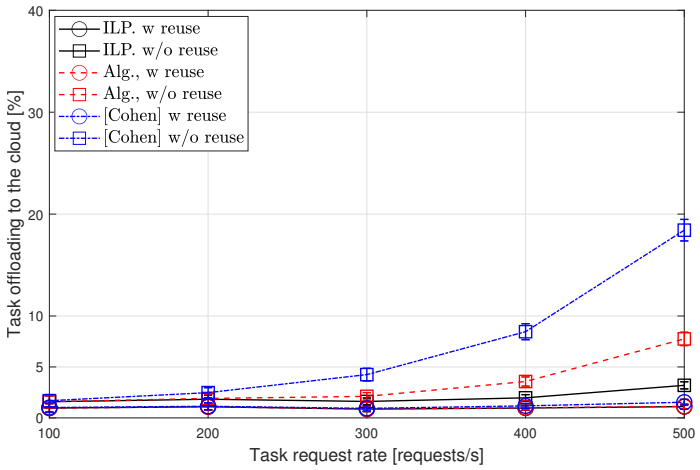
Heuristics validation

In this section, we compare the performance of our heuristic method (curves labeled as *Alg.*. in Figures) to the proposed ideal placement strategy (curves labeled as *ILP* in Figures) when computing task requests are varied from 100 to 500.

Results are shown for both the scenario in which the output of the computing tasks is reused (curves labeled as *with reuse*) and the scenario in which it is not (curves labeled as



(a) Exchanged data.

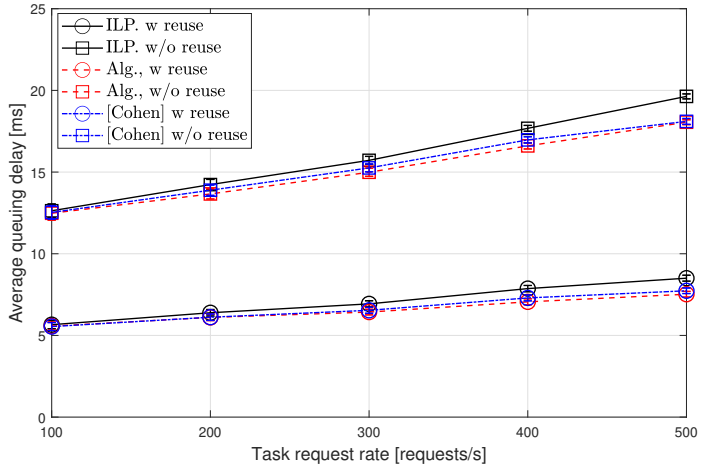


(b) Task offloading to the cloud

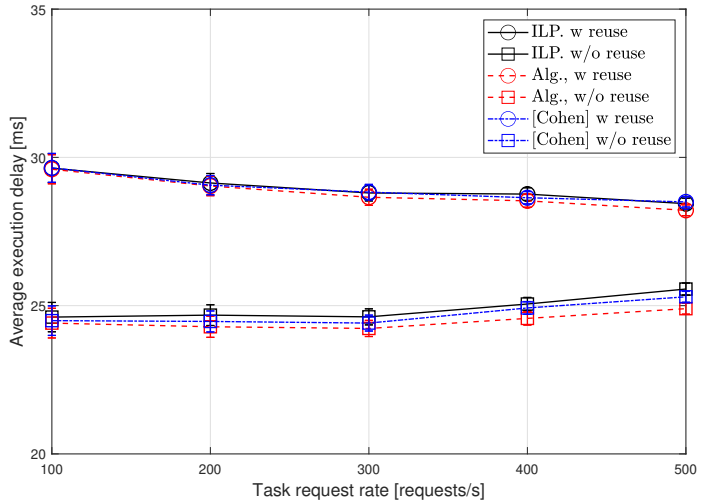
Fig. 5.2. Heuristics validation: effectiveness metrics (α=0.8, $\overline{T}_j^{max}$ = 300 ms).

with/without reuse). It can be shown that both of the heuristics under consideration closely approximate the ideal answer. In particular, when compared to the benchmark, our heuristic comes closer to the ILP results.

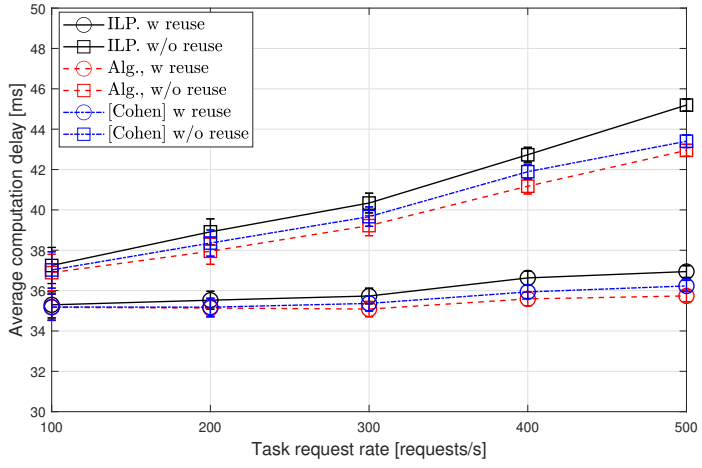
For the purpose of validation, the metrics in Figs. 5.2 and 5.3 are taken into account while setting the Zipf distribution's skewness parameter α to 0.8. The volume of data transferred within the edge domain is depicted in Fig. 5.2(a). Due to the greater amount of input data that needs to be transferred, it makes sense for both solutions that are being evaluated that the metric rises as the rate of requests does. Furthermore, reuse results in a nearly a one order of magnitude reduction in the amount of data traffic exchanged. Unsurprisingly, the ILP offers the lowest values for the metric of transferred data that is taken into consideration. The ILP is more successful in locating the ideal work placement solution by investigating several options, which ensures that the goal function is minimized.



(a) Average queuing delay.



(b) Average execution delay.



(c) Average computation delay.

Fig. 5.3. Heuristics validation: delay metrics (α=0.8, $\overline{T}_j^{max}$ = 300 ms).

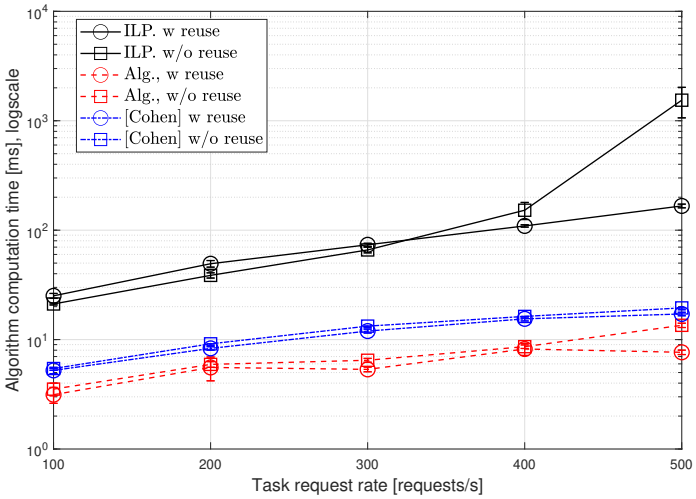


Fig. 5.4. Heuristics Vs. ILP: algorithm computation delay.

Some of the requested tasks are offloaded to the cloud (Fig. 5.2(b)) because not all edge nodes, after they have been loaded, are capable of processing the requested tasks ensuring the computation delay constraint in Eq. (5.12). Unsurprisingly, as the task request rate rises, this happens with a high likelihood when task reuse is not possible. Furthermore, our heuristic has a smaller percentage, demonstrating its superiority to the benchmark one.

Indeed, our algorithm populates each node by greedily taking only those tasks from the list of tasks (line 18, see Algorithm 1) that have chosen that node as their best executor. Contrarily, in the benchmark under consideration, according to Cohen [123], each node is greedily filled by accepting from the whole list of tasks to be allocated, regardless of whether the node under consideration is the most appropriate executor for those tasks. Non-optimal assignments may happen there after a few iterations. They are then resolved by shifting a task that was previously given to one node to another that has a lower cost, as determined by the computation of marginal cost. The node where the task is unallocated may waste some resources as a result of this workaround.

As seen in Fig. 5.3(b), the reduced amount of exchanged data traffic in the event of reuse is compensated for by increased execution delays. This is because less capable edge nodes (instead of the distant cloud) are chosen as executors for some tasks in the case of reuse. On the other hand, because some tasks can be reused for a time and are not required to be processed, the queuing delays (Fig. 5.3(a)) are smaller. As a result, calculation delay values (Fig. 5.3(c)) in the reuse scenario are lower than those in the scenario where each request is handled by a fresh task execution.

As a further result, in Fig. 5.4 we report the time required to run the heuristic algorithms as well as to solve the ILP through the standard solver. In both cases, with and without the reuse of the output, the computation times of the two heuristics are significantly lower than those measured for the ILP solution, reasonably confirming their greater efficiency. They are both well below 20 ms, with our heuristic which is slightly faster. Hence, in the

following, we will report results only for it. Moreover, the proposed heuristic shows even shorter computation times when reuse is enabled: fewer alternatives need to be explored compared to the case in which reuse is not enabled and the task placement solution can be found more quickly. Hence, in the following, we will refer only to it.

As a further result, we present the time needed to run the heuristic algorithms and solve the ILP using the conventional solver in Fig. 5.4. The computation times of the two heuristics are much shorter than those recorded for the ILP solution in both situations - with and without the reuse of the output - which logically confirms their superior efficiency. Both of these are considerably below 20 ms, with our heuristic that is a little bit quicker. In addition, the proposed heuristic demonstrates even faster computation times when reuse is enabled because there are less options to consider and a quicker time to find the optimal task placement.

Hence, we will just present the results for it in the sections that follow.

Our proposal Vs. benchmark placement solutions

The proposed placement strategy's results are contrasted with two benchmark solutions. The first one is an example of a traditional MEC approach (labeled as *MEC w cloud* in the figures), in which tasks are only *in purpose-built servers* that are only connected to access nodes in the edge domain and offloaded to the cloud whenever delay constraints cannot be met because of the depletion of those servers' resources. The second one (referred to as *onlyCloud* in the figures) deals with the scenario in which *all tasks are executed in the remote cloud*.

Impact of tasks popularity

The first simulation campaign is run to assess how much the popularity of the tasks affects the performance. We contrast the results obtained for the Zipf distribution, for skewness parameter α equal to 0.8 and 1.2 (curves labeled as *Zipf* in Figures), with those obtained in the presence of a uniform task request distribution (curves labeled as *Uniform* in Figures). The campaign is run with task requests varied from 100 to 2000 requests per second. The more popular the tasks are (i.e., the higher α), the better the performances. Indeed, lower data is exchanged in the domain (Fig. 5.5(a)), a lower number of tasks are offloaded to the cloud (Fig. 5.5(b)), a higher edge computation hit (Fig. 5.5(c)), and a lower computation delay (Fig. 5.6), when the $\alpha = 1.2$. This is because requests focus on a small number of tasks, and after these are processed, their output can be reused to serve multiple consumers, reducing the processing load on edge nodes and eliminating the requirement for traffic exchange to get the input data.

Results also demonstrate that our proposal experiences a notable reduction in traffic traversing the edge domain (greater reduction as the popularity increases) in comparison to the *MEC w cloud* solution. When $\alpha = 1.2$ is used and when task request rate is 2000

requests/s, the reduction is up to about 91%. Our proposal, which distributes tasks across the entire edge domain, is substantially more effective than limiting task execution to purpose-built edge servers. No matter how popular a task is thought to be, edge servers connected to leaf nodes are compelled to offload at least half of it to the cloud (Fig. 5.5(b)). Locally processed tasks have a significant queuing time, which increases the total computation delay (Fig. 5.6)[4].

For $\alpha = 1.2$ and a task request rate of 2000 requests/s, gains in terms of decreased exchanged traffic compared to the baseline *onlyCloud* approach are in the order of 93%

Fig. 5.5(c) demonstrates that, on average, more than half of the outputs are reused to address several requests (up to 85% in the case of α=1.2 and 2000 requests/s).

To clarify the effectiveness of the suggested task allocation technique even more, we calculate the average computation latency metric as a function of the normalized edge resource usage (NRU), which is outlined as:

$$NRU = \frac{\sum_{j \in M} \lambda_j^{exec} \cdot l_j}{\sum_{i \in N} \mu_i}. \tag{5.16}$$

In Eq. (5.16), the denominator indicates the total number of CPU cycles per second that are delivered by the edge nodes, while the numerator represents the effective CPU cycles per second needed to process the requested tasks. Therefore, NRU provides a measurement of how the computing load that is being offered compares to the total computing resources that are available in the edge domain. It is worth to underline that the cloud's resources were purposefully left out of the denominator of Eq. (5.16).

The simulations were run with a variable number of requested tasks in the range of $[100, 2000]$, with $\overline{T}_j^{max} = 500ms$ and the remaining parameters set in accordance with Table 5.2. The average computation delay vs the normalized resource utilization for each simulation has been calculated and is shown as a scatterplot in Fig. 5.7(a). It can be seen that, due to increasing reuse, edge resource usage decreases as popularity (thus α) increases.

Additionally, it can be seen that as resource usage rises, the scattered points for the *MEC w cloud* benchmark have a sharper slope in comparison to the proposal approach. This is due to the rapid depletion of the processing resources at the bottom of the edge topology.

In order to better reflect a network operator-centric perspective, we derive the extra-domain rate (EDR) as a function of normalized edge resource (Fig. 5.7(b)).

It is derived as follows:

$$EDR = \sum_{j \in M} X_{d,j} \lambda_j^{exec} \cdot S_{d,j}. \tag{5.17}$$

For the *cloudOnly* solution and in a comparable way, albeit less dramatically, for the *MEC w cloud* benchmark, the metric grows linearly as the normalized edge resource utilization rises. It can be observed that the values for this metric are more than halved when compared to the case in which all tasks are executed in the cloud, by saving precious bandwidth resources.

[4] Results are not reported for the *cloudOnly* solution, since the delay is negligible thanks to the virtually unlimited capabilities of the cloud.

Impact of the output time validity

The second simulation campaign is run both not enabling the compute reuse and enabling it varying the mean time validity of the reusable output of computing tasks ($\overline{T}_j^{max} = 0.5$ s and 1 s).

No matter the solution under consideration, the measured performances are better the higher the time validity, as shown in Figs. 5.8 and 5.9. Additionally, with these settings, the proposal's domain traffic exchange is considerably less than that of the benchmark solutions (Fig. 5.8(a)).

The benefits of a longer time validity become even more striking when the frequency of task requests rises. In specifically, for $\overline{T}_j^{max}$ equal to 1 s and task request rate equal to 2000 requests/s, it is decreased up to 71.2% and 82.9%, respectively, in comparison to the *MEC w cloud* and *onlyCloud* solutions.

When reuse is not possible, the edge domain cannot handle the demand and a significant portion of the tasks are offloaded to the cloud, with more than half of them offloaded above 1500 requests/s (Fig. 5.8(b)). Already for an offered load of 100 requests/s, the *MEC w cloud* benchmark method offloads more than 60% of task requests. The increased computation delay, shown in Fig. 5.9, which is mostly caused by the higher queuing delay contribution, further supports the higher load in the absence of reuse.

5.8 Main findings

The deployment of computing tasks with time-limited reusable outputs over an SDN-based network edge domain has been the subject of our strategy in this chapter. An ILP problem is used to develop the optimal solution, which aims to minimize the use of network usage when choosing the candidate executors. The latter ones includes SDN edge nodes and a remote cloud as the ultimate choice, avoiding edge node saturation and limiting task computation delay.

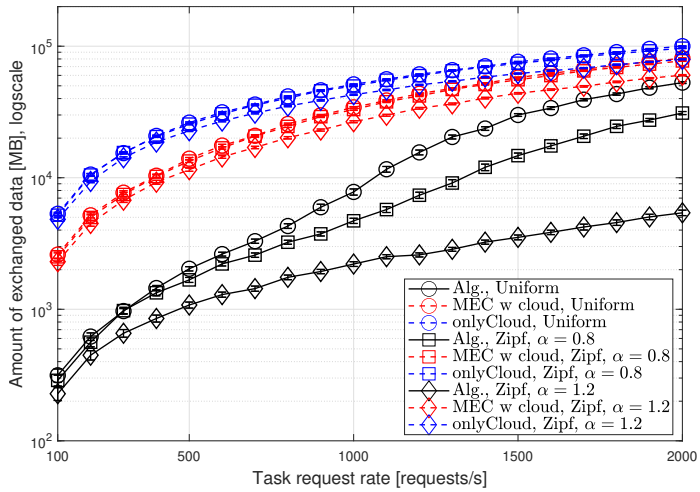
The decision is shown to be implemented in an affordable manner using a novel heuristic algorithm that has been proposed. It offers a near-optimal placement solution by well approximating the one offered by the ILP. When reuse is enabled, which is the setting we are most interested in, the heuristics approximates better the optimal solution. The formulated optimization problem is also solved thanks to a heuristic proposed in the literature with a similar efficiency in terms of algorithm execution time.

A large-scale simulation campaign has been run to evaluate the behavior of our approach in comparison to existing placement strategies using a series of useful metrics under various task requests, task popularity distributions, and time validities of the computing task's output. Results indicate that the approach effectively satisfies the task requirements and efficiently utilizes network edge resources while offloading a minimal number of tasks to the cloud. Benefits are especially noticeable when popular tasks are taken into account (greater

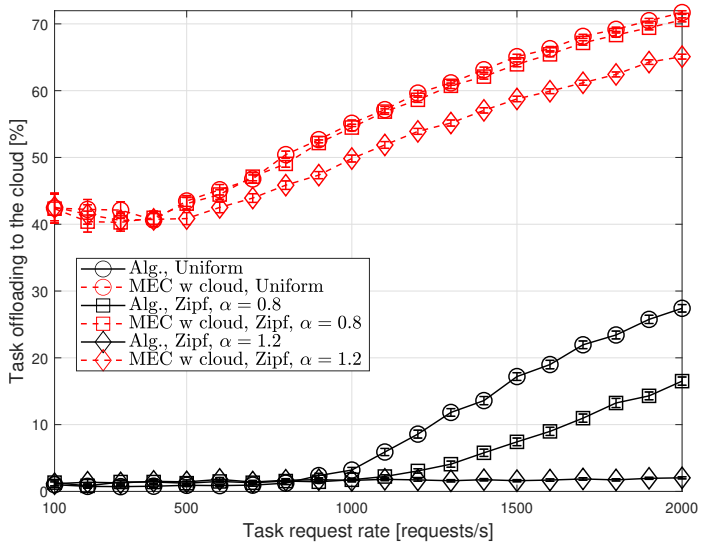
Zipf skewness parameter) with a longer time validity of the task output, despite the rising of task requests. Interestingly, our idea takes down the amount of traffic exchanged in the edge domain by up to 93% when compared to the default *onlyCloud* approach.

The achieved results open the road for enabling the reuse of the computing tasks' output in order to maximize the potential of the network edge resources.

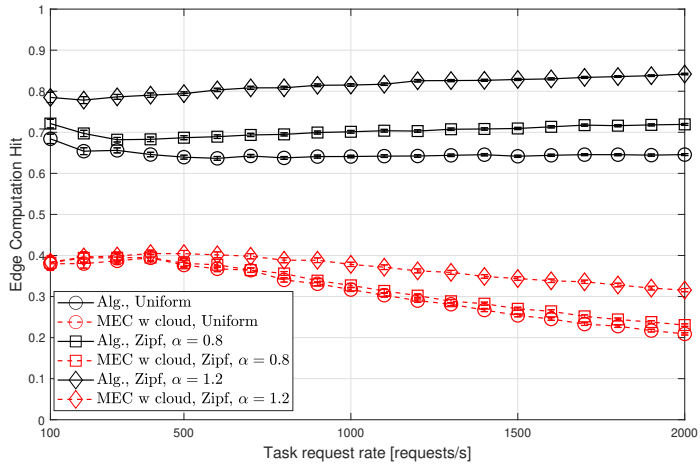
However, as explained in the conclusions of the thesis (see next Chapter 6), a series of challenges still need to be faced and necessitate additional research.



(a) Exchanged data.



(b) Task offloading to the cloud



(c) Edge computation hit

Fig. 5.5. Effectiveness metrics when varying tasks popularity ($\overline{T}_j^{max} = 500\ ms$).

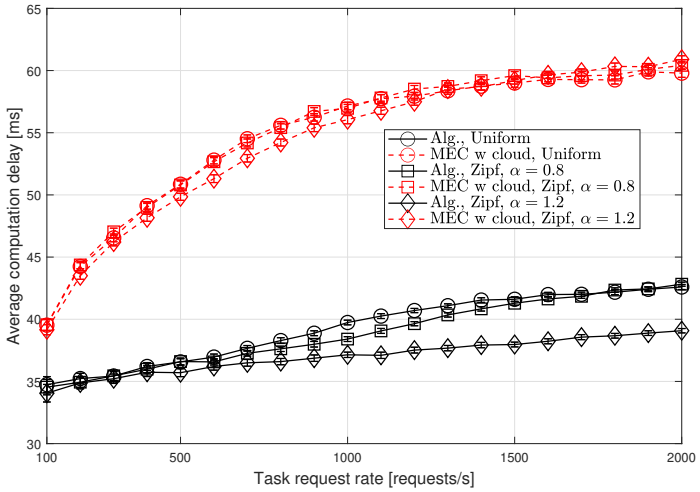
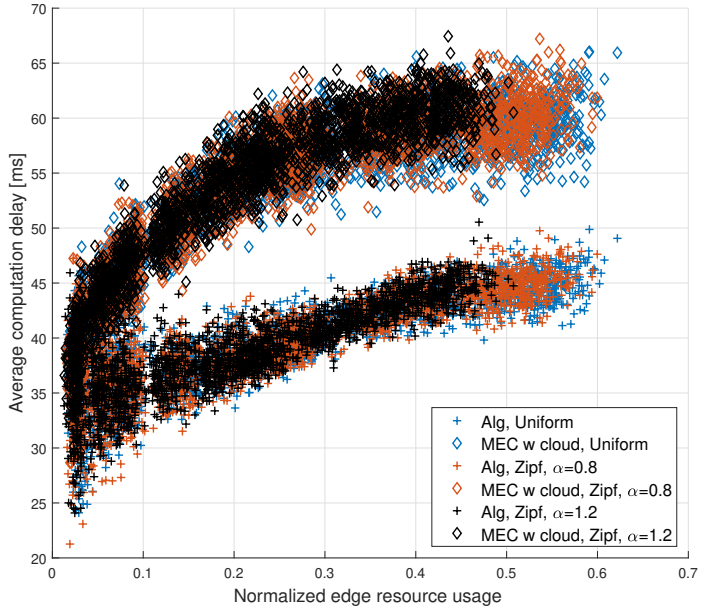
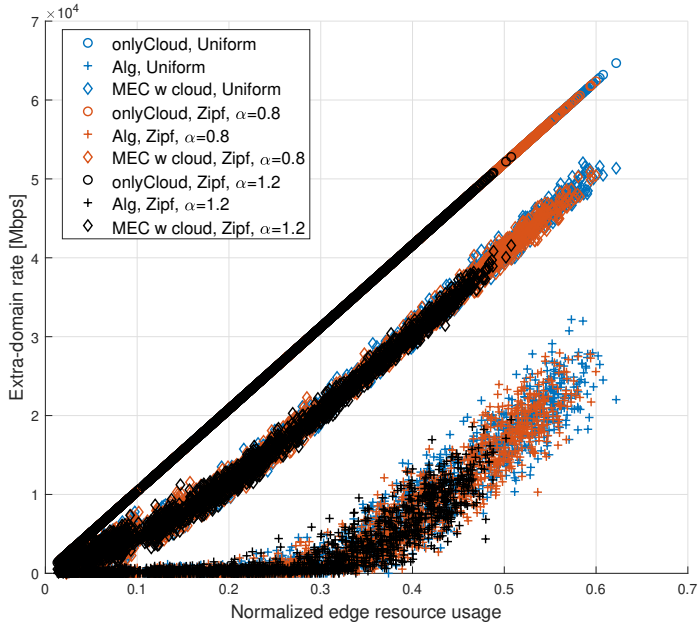


Fig. 5.6. Average computation delay when varying tasks popularity ($\overline{T}_j^{max} = 500\ ms$).

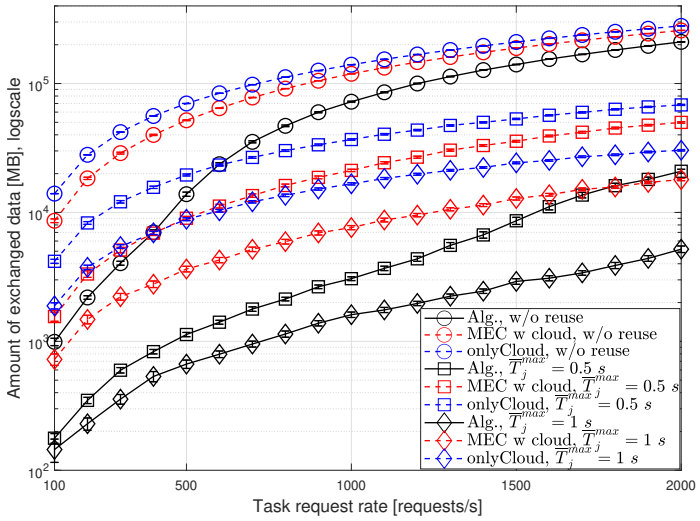


(a) Average computation delay

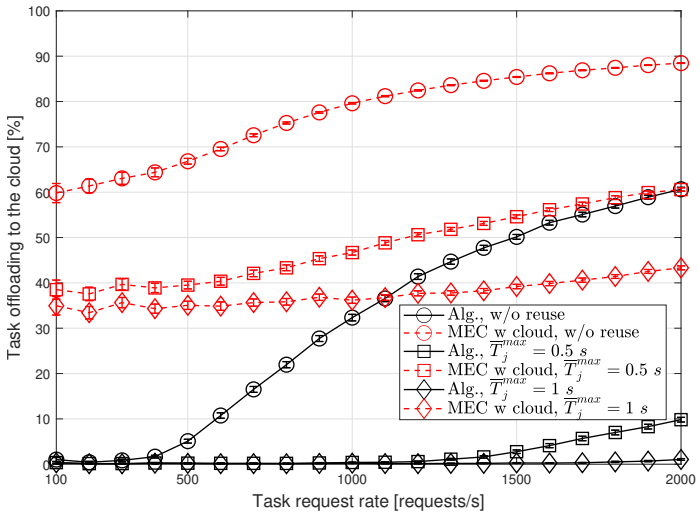


(b) Extra-domain rate

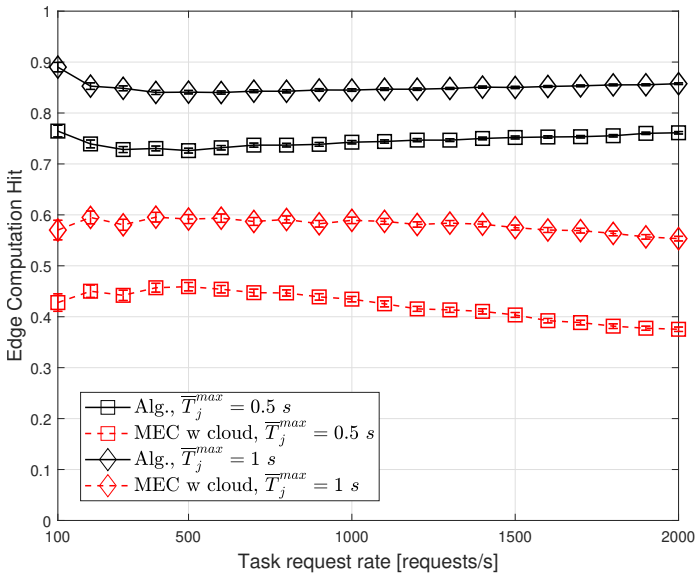
Fig. 5.7. Metrics vs normalized edge resource usage ($\overline{T}_j^{max} = 500\ ms$).



(a) Exchanged data.



(b) Task offloading to the cloud



(c) Edge computation hit

Fig. 5.8. Effectiveness metrics when varying the time validity of the output ($\alpha = 0.8$).

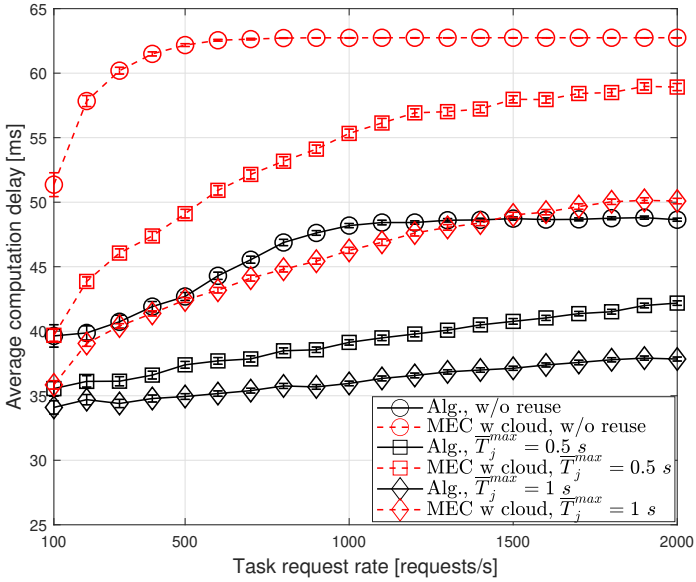


Fig. 5.9. Average computation delay when varying the time validity of the output ($\alpha = 0.8$).

6

Conclusions and future works

Several efforts have been made in the research community as well as by industrial companies, telco operators, and standardization bodies, to make it possible the provisioning of new 5G/6G services. The latter ones encompass applications like augmented and extended reality, automotive, low latency gaming, industry 4.0 applications that can offer different life styles to the society, spanning from industries' workflows to the hobbies.

In such a context, the evolution from cloud computing to edge computing paradigm plays a crucial role. The edge computing paradigm aims to move storage and processing resources from the cloud to the network nodes closest to the user, the so-called edge nodes. This new approach makes such resources available, thus supporting the connectivity requirements (low latency, high throughput) of the required services. Not only that, it also allows decreasing the amount of data transmitted in the core network, as requests are fulfilled near the access segment ensuring low congestion.

Moreover, although it is still in a early research stage, the new concept of edge intelligence is considered as a strong paradigm for the next future 6G systems to provide new kind of services.

Albeit standardization institutes and researchers have made greats steps ahead about this topic, some issues related to the edge computing paradigm have to be faced. Mobility awareness, task offloading decision, and privacy are listed as some of the open issues identified in ETSI specifications [14, 20, 21].

In this thesis the task offloading decision issue has been faced. In particular a new strategy for placing *CPU-intensive* and *delay-constrained* reusable computing tasks into a softwarized edge domain has been designed. Within the thesis, the proposed strategy has been formulated as an ILP problem aimed at ensuring that the delay constraints of each computing task are met, but also at minimizing the network resources usage by reducing the amount of data exchanged within the edge domain.

It was proved that the formulated problem is an NP-hard problem. It was solved at first with optimal standard solvers. Despite the first obtained solution is the one optimally minimizing the objective function, due to the fact it is an NP-hard problem it carries feasibility and scalability issues.

Thus, to face these issues, two kind of methodologies have been proposed to solve the formulated problem even at the cost of sub-optimal results. On one hand, a series of ML-based solutions have been designed, implemented, and tested, while evaluating not only network- and task-, but also ML-related metrics.

On the other hand, also a heuristic has been implemented and tested.

Furthermore, for all near-optimal proposed solutions their feasibility, scalability, and better performance against every consider benchmark approach has been proven.

If we would compare the two considered solutions to efficiently solve the optimization problem, the heuristic algorithm seems to be the best approach to follow. In fact, while the heuristic algorithm natively guarantees that each constraint of the formulated problem is met, a ML-based solution may incur into misclassification errors. Moreover, in high dynamic context, supervised learning may suffer of the lack of labeled data. Even though the former issue can be overstep, there is still the necessity to perform the training phase. However, their thorough comparison, also in terms of energy consumption will be a subject matter of future works.

6.1 Future Research

The encouraging results achieved in this thesis pave the way for further investigation.

From a general perspective, the mobility awareness and security issues (explained in Section 2.7) have not been faced in the thesis's work. Of course they are challenging aspects in which researchers are working on and their findings can enhance as well as complement the work of this thesis. Instead, for what concerns the straightforward future works for the proposed solutions they are explained in the following.

Caching of computation's result. In the thesis we focused on the use cases in which, the result of computation tasks are quite low in terms of size. In these cases, thus under the assumption about the computation's results it has been made in the study, there is no the necessity to remove some results in favor of others because it is assumed that the storage capabilities of the edge nodes are large enough to store all the results of the computations. Moreover, this is further motivated by the fact that these results have a limited lifetime and thus should be removed regardless. Therefore, this work could be improved by considering different types of applications, which have computational results with more significant storage requirements.

The need of labeled data for machine learning. There is still opportunity for improvement in the research. Task offloading decisions may encounter unexpected situations in highly dynamic edge environments, such as task request rates, for which pertinent labeled data (necessary for supervised learning approach) may be difficult to gather. This problem can be resolved using semi-supervised learning (SSL) methods, which would enhance our supervised learning solutions. Among the numerous SSL techniques that have been thoroughly

studied, for example in [126], wrapper methods can be combined with the ML techniques that have been shown to be the best performing, DT and MLP. Of course, more research is needed to determine whether these SSL techniques, which can be applied to almost any supervised base learner, are applicable in our environment so that we can fully comprehend the performance gains and the cost associated with achieving them. For instance, self-training, the most fundamental kind of pseudo-labelling, would need more time because iterations are needed in order to acquire predictions for the unlabeled data samples in addition to training the model on labelled data.

Objective function. While maintaining the advantages of time-limited compute reuse, the proposed framework can be adjusted to take into account the optimization of various objectives, such as the network latency experienced in data interchange.

Energy-aware task placement solutions. All around the globe, emerged a strong attention toward environmental aspects which led to a strong push toward a reduction of the carbon footprint of almost every human activity not excluding ICT aspects. As human beings are becoming increasingly dependent on technology, the relationship of ICT with the natural environment is continuously degrading. In the past, ICT was often regarded as a low-carbon enabler, but its widespread adoption has adversely turned it into a power drainer. The negative consequence is climate change, which is a complex social problem with rebound effects on human beings. Taking cognizance of this problem becomes vital for each and every computer user, telco companies, and not only, to contribute and reduce the growing carbon footprint of ICT. New research activities could contribute to research in this field with the designing of a "sustainable" framework for a joint orchestration of computational and network resources at the edge. Specifically, the carbon footprint of the optimization algorithms could be taken into account, as those algorithms are usually computation intensive and require some signalling overhead. The research challenge would be if makes sense to design a "sort of carbon aware" heuristic even at the cost of trading off some lower to the optimum for a lower carbon footprint. Of course, this question is not trivial to answer, it largely depends on the task at end, and requires a knowledge of how the single operation required to solve a heuristic impact on the power consumption of the machine executing the algorithm.

Practical deployment. Edge nodes should be able to distinguish between several requests for the same task in order to enable compute reuse. It is possible to achieve this by utilizing either *(i)* application layer solutions, such as Deep Packet Inspection (DPI) or proxy-based mechanisms [127], or *(ii)* information-centric Named Data Networking (NDN) architecture, which uses names directly at the network layer to uniquely identify contents as well as generic tasks/services [128], [129]. The former solutions incur a greater overhead. The latter option simply requires a lookup operation in the NDN tables during the request forwarding process, which normally imposes a minor latency if names are properly encoded [130]. Practically speaking, this would mean that customers, SDN Controllers, and potential task executors would need to agree on the definition of semantically rich task names.

Time validity estimation. In the work, the assumption has been made that the output of

the computing task would be valid for a particular task request instance over time. It may differ from one task instance to the next. However, in some cases, the real parameter may be shorter than what was initially set (if the environment's conditions, such as the input data, unexpectedly change), which would render the computing task's output invalid. To accurately forecast the time when the calculation output is no longer valid, artificial intelligence technologies could be implemented at the executor. Alternately, as envisioned in the proposed architecture, the input data source(s) might determine the validity time and communicate it to the executor along with some additional cache control directives. To specify that task results must be re-validated with the input data source(s), respectively, upon each reuse and at the estimated expiration, information like "no-cache" and "must-revalidate" directives, which are currently leveraged by the Hyper Text Transfer Protocol (HTTP) cache control mechanism, could be introduced in our design. In order to balance the accuracy of the results with the incurred latency and signaling traffic introduced into the network, more research is required.

References

1. M. Series, "Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond," *Recommendation ITU*, pp. 2083–0, 2015.
2. F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "Nist cloud computing reference architecture," *NIST special publication*, vol. 500, no. 2011, pp. 1–28, 2011.
3. "Microsoft Azure," `https://azure.microsoft.com/it-it/overview/what-is-paas/`.
4. H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
5. "Mobile Edge Computing - Introductory Technical White Paper, 2014," `https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf`.
6. F. Giust, X. Costa-Perez, and A. Reznik, "Multi-access edge computing: An overview of etsi mec isg," *IEEE 5G Tech Focus*, vol. 1, no. 4, p. 4, 2017.
7. S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.
8. X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.
9. "Software-Defined Networking: The New Norm for Networks, 2012," `https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/`.
10. W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The road towards 6g: A comprehensive survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.
11. C. Han, Y. Wu, Z. Chen, *et al.*, "Network 2030 a blueprint of technology applications and market drivers towards the year 2030 and beyond," 2018.

12. S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
13. M. Maray and J. Shuja, "Computation offloading in mobile cloud computing and mobile edge computing: survey, taxonomy, and open issues," *Mobile Information Systems*, vol. 2022, 2022.
14. "ETSI GS MEC 003 V2.1.1 (2018-10," `https://www.etsi.org/deliver/etsi_gs/mec/001_099/002/02.01.01_60/gs_mec002v020101p.pdf`.
15. E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks, *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.
16. B. Li, Z. Fei, and Y. Zhang, "Uav communications for 5g and beyond: Recent advances and future trends," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2241–2263, 2018.
17. M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.
18. S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2017.
19. J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 181, p. 103005, 2021.
20. "ETSI GR MEC 018 V1.1.1 (2017-10," `https://www.etsi.org/deliver/etsi_gr/mec/001_099/018/01.01.01_60/gr_mec018v010101p.pdf`.
21. "Technical Report ETSI White Paper No. 46. MEC security: Status of standards support and future evolutions. ," `https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP-46-2nd-Ed-MEC-security.pdf`.
22. Z. Xiao and Y. Xiao, "Security and privacy in cloud computing," *IEEE communications surveys & tutorials*, vol. 15, no. 2, pp. 843–859, 2012.
23. H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, p. 102781, 2020.
24. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
25. I. Kunze, K. Wehrle, D. Trossen, and M.-J. Montpetit, "Use cases for in-network computing," *IETF, Internet-Draft*, 2021.
26. M. McBride *et al.*, "Edge Data Discovery for COIN, IETF Computing in the Network Research Group," November 2020.

27. B. Nour, S. Mastorakis, and A. Mtibaa, "Compute-less networking: Perspectives, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 259–265, 2020.
28. M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, and A. Iera, "SDN-managed provisioning of named computing services in edge infrastructures," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1464–1478, 2019.
29. A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Comm. Letters*, vol. 6, no. 3, pp. 398–401, 2017.
30. Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.
31. M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.
32. G. Lia, M. Amadeo, C. Campolo, G. Ruggeri, and 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.
33. A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.
34. M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.
35. S. Goudarzi, M. H. Anisi, H. Ahmadi, and L. Musavian, "Dynamic resource allocation model for distribution operations using SDN," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 976–988, 2020.
36. S. Choo, J. Kim, and S. Pack, "Optimal task offloading and resource allocation in software-defined vehicular edge computing," in *Int. Conf. on information and communication technology convergence (ICTC)*, pp. 251–256, 2018.
37. Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2019.
38. S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.
39. T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. on Vehi. Technology*, vol. 68, no. 1, pp. 856–868, 2018.
40. S. Jošilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 667–680, 2020.

41. J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24290–24307, 2019.
42. Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *2017 IEEE wireless communications and networking conference (WCNC)*, pp. 1–6, IEEE, 2017.
43. Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Generation Computer Systems*, vol. 112, pp. 148–161, 2020.
44. S.-W. Ko, K. Han, and K. Huang, "Wireless networks for mobile edge computing: Spatial modeling and latency analysis," *IEEE Trans. on Wireless Comm.*, vol. 17, no. 8, pp. 5225–5240, 2018.
45. Z. Li, V. Chang, H. Hu, D. Yu, J. Ge, and B. Huang, "Profit maximization for security-aware task offloading in edge-cloud environment," *Journal of Parallel and Distributed Computing*, 2021.
46. T. Zhao, S. Zhou, X. Guo, and Z. Niu, "Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *IEEE ICC*, pp. 1–7.
47. C. Jiang, Y. Chen, Q. Wang, and K. R. Liu, "Data-driven stochastic scheduling and dynamic auction in IaaS," in *IEEE GLOBECOM*, pp. 1–6.
48. M. Zukerman, "Introduction to queueing theory and stochastic teletraffic models," *arXiv preprint arXiv:1307.2968*, 2013.
49. T. Subramanya, D. Harutyunyan, and R. Riggio, "Machine learning-driven service function chain placement and scaling in MEC-enabled 5G networks," *Computer Networks*, vol. 166, p. 106980, 2020.
50. M. Adhikari, M. Mukherjee, and S. N. Srirama, "DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5773–5782, 2019.
51. J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2010.
52. P. Van Mieghem, *Performance analysis of communications networks and systems*. Cambridge University Press, 2009.
53. X. Hao, R. Zhao, T. Yang, Y. Hu, B. Hu, and Y. Qiu, "A risk-sensitive task offloading strategy for edge computing in industrial internet of things," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–18, 2021.
54. J. Yang, B. Ai, I. You, M. Imran, L. Wang, K. Guan, D. He, Z. Zhong, and W. Keusgen, "Ultra-reliable communications for industrial internet of things: Design considerations and channel modeling," *IEEE Network*, vol. 33, no. 4, pp. 104–111, 2019.
55. M. Masoudi and C. Cavdar, "Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption," *IEEE Transactions on Mobile Computing*, vol. 20, no. 12, pp. 3324–3337, 2020.

56. L. Leng, J. Li, H. Shi, and Y. Zhu, "Graph convolutional network-based reinforcement learning for tasks offloading in multi-access edge computing," *Multimedia Tools and Applications*, vol. 80, no. 19, pp. 29163–29175, 2021.

57. L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019.

58. J. Almutairi, M. Aldossary, H. A. Alharbi, B. A. Yosuf, and J. M. Elmirghani, "Delay-optimal task offloading for uav-enabled edge-cloud computing systems," *IEEE Access*, 2022.

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

60. A. Ali-Eldin, B. Wang, and 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*, pp. 1–12, 2021.

61. D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European journal of operational research*, vol. 60, no. 3, pp. 260–272, 1992.

62. S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

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

64. T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 38–67, 2019.

65. H. Guo, J. Liu, and J. Lv, "Toward intelligent task offloading at the edge," *IEEE Network*, vol. 34, no. 2, pp. 128–134, 2019.

66. X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

67. A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, vol. 182, p. 107496, 2020.

68. J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

69. S. Yu, X. Wang, and 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)*, pp. 1–6, 2017.

70. S. M. Araújo, F. S. de Souza, and G. R. Mateus, "A hybrid optimization-Machine Learning approach for the VNF placement and chaining problem," *Computer Networks*, vol. 199, p. 108474, 2021.

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

72. D. Rahbari and M. Nickray, "Task offloading in mobile fog computing by classification and regression tree," *Peer-to-Peer Networking and Applications*, vol. 13, no. 1, pp. 104–122, 2020.

73. S. Wu, W. Xia, W. Cui, Q. Chao, Z. Lan, F. Yan, and 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)*, pp. 1–6, 2018.

74. C. Sonmez, C. Tunca, A. Ozgovde, and C. Ersoy, "Machine learning-based workload orchestrator for vehicular edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 4, pp. 2239–2251, 2020.

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

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

77. N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

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

79. J. Zhang, H. Guo, and J. Liu, "Adaptive task offloading in vehicular edge computing networks: a reinforcement learning based scheme," *Mobile Networks and Applications*, vol. 25, no. 5, pp. 1736–1745, 2020.

80. N. Kiran, C. Pan, and 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)*, pp. 268–273, IEEE, 2020.

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

82. G. Lia, M. Amadeo, G. Ruggeri, C. Campolo, A. Molinaro, and V. Loscrì, "In-network placement of delay-constrained computing tasks in a softwarized intelligent edge," *Computer Networks*, p. 109432, 2022.
83. G. Aurélien, "Hands-on machine learning with scikit-learn & tensorflow," *Geron Aurelien*, 2017.
84. H. Huang and H. V. Burton, "Classification of in-plane failure modes for reinforced concrete frames with infills using machine learning," *Journal of Building Engineering*, vol. 25, p. 100767, 2019.
85. P. C. Sen, M. Hajra, and M. Ghosh, "Supervised classification algorithms in machine learning: A survey and review," in *Emerging technology in modelling and graphics*, pp. 99–111, Springer, 2020.
86. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.
87. J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.
88. L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.
89. M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks*, vol. 6, no. 4, pp. 525–533, 1993.
90. 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, p. 6, 2021.
91. S. Haykin, "Neural networks: a comprehensive foundation," *The Knowledge Engineering Review*, vol. 13, no. 4, pp. 409–412, 1999.
92. E. Mayoraz and E. Alpaydin, "Support vector machines for multi-class classification," in *International Work-Conference on Artificial Neural Networks*, pp. 833–842, Springer, 1999.
93. S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2019.
94. K. Toczé and S. Nadjm-Tehrani, "Orch: Distributed orchestration framework using mobile edge devices," in *IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, 2019.
95. Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.
96. H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

97. I. Markoulidakis, I. Rallis, I. Georgoulas, G. Kopsiaftis, A. Doulamis, and N. Doulamis, "Multiclass confusion matrix reduction method and its application on net promoter score classification problem," *Technologies*, vol. 9, no. 4, p. 81, 2021.

98. M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

99. M. Naser and A. H. Alavi, "Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences," *Architecture, Structures and Construction*, pp. 1–19, 2021.

100. S. Shannigrahi, S. Mastorakis, and F. R. Ortega, "Next-generation networking and edge computing for mixed reality real-time interactive systems," in *IEEE ICC Workshops 2020*, pp. 1–6.

101. S. Bi, L. Huang, and Y.-J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.

102. J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM*, pp. 207–215, 2018.

103. J. Lee, A. Mtibaa, and S. Mastorakis, "A case for compute reuse in future edge systems: An empirical study," in *IEEE GLOBECOM Workshops*, pp. 1–6, 2019.

104. S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, "ICedge: when Edge Computing meets Information-Centric Networking," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.

105. M. W. Al Azad and S. Mastorakis, "Reservoir: Named data for pervasive computation reuse at the network edge," in *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 141–151, IEEE, 2022.

106. B. Ottenwälder, B. Koldehofe, K. Rothermel, K. Hong, and U. Ramachandran, "Recep: Selection-based reuse for distributed complex event processing," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 59–70, 2014.

107. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Comm. Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

108. X.-Q. Pham, T.-D. Nguyen, E.-N. Huh, *et al.*, "Joint service caching and task offloading in multi-access edge computing: A QoE-based utility optimization approach," *IEEE Communications Letters*, 2020.

109. M. Kim, H. Cho, Y. Cui, and J. Lee, "Service caching and computation resource allocation for large-scale edge computing-enabled networks," in *IEEE GLOBECOM*, pp. 1–6, 2020.

110. P. Liu, G. Xu, K. Yang, K. Wang, and X. Meng, "Jointly optimized energy-minimal resource allocation in cache-enhanced mobile edge computing systems," *IEEE Access*, vol. 7, pp. 3336–3347, 2018.

111. G. Lee, W. Saad, and M. Bennis, "Online optimization for low-latency computational caching in fog networks," in *IEEE Fog World Congress (FWC) 2017*, pp. 1–6.

112. Y. Cui, W. He, C. Ni, C. Guo, and Z. Liu, "Energy-efficient resource allocation for cache-assisted mobile edge computing," in *IEEE Conf. on Local Computer Networks (LCN)*, pp. 640–648, 2017.

113. P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in *Int. Conf. on Mobile Computing and Networking*, pp. 19–34, 2018.

114. R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.

115. T. N. B. Duong and S. Zhou, "A dynamic load sharing algorithm for massively multiplayer online games," in *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, pp. 131–136, IEEE, 2003.

116. L. Chen, J. Xu, and S. Zhou, "Computation peer offloading in mobile edge computing with energy budgets," in *IEEE GLOBECOM*, pp. 1–6, 2017.

117. S. Vural, N. Wang, P. Navaratnam, and R. Tafazolli, "Caching transient data in internet content routers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1048–1061, 2016.

118. P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1300–1313, 2018.

119. M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.

120. S. Lu, J. Wu, Y. Duan, N. Wang, and J. Fang, "Towards cost-efficient resource provisioning with multiple mobile users in fog computing," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 96–106, 2020.

121. K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

122. M. S. Elbamby, M. Bennis, and W. Saad, "Proactive edge computing in latency-constrained fog networks," in *EuCNC 2017*, pp. 1–6.

123. R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.

124. T. Zhu, J. Li, Z. Cai, Y. Li, and H. Gao, "Computation scheduling for wireless powered mobile edge computing networks," in *IEEE INFOCOM 2020*, pp. 596–605.

125. A. Mukhopadhyay, G. Iosifidis, and M. Ruffini, "Migration-aware network services with edge computing," *IEEE Transactions on Network and Service Management*, 2021.

126. J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

127. Y. Cui, J. Song, M. Li, Q. Ren, Y. Zhang, and X. Cai, "SDN-based big data caching in ISP networks," *IEEE Transactions on Big Data*, vol. 4, no. 3, pp. 356–367, 2017.

128. M. Król and I. Psaras, "Nfaas: named function as a service," in *ACM Conf. on Information-Centric Networking*, pp. 134–144, 2017.

129. M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "IoT services allocation at the edge via named data networking: From optimal bounds to practical design," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 661–674, 2019.

130. D. He, D. Zhang, K. Xu, K. Huang, and Y. Li, "A fast and memory-efficient approach to NDN name lookup," *China Communications*, vol. 14, no. 10, pp. 61–69, 2017.