network

Article

Mobility- and Energy-Aware Cooperative Edge Offloading for Dependent Computation Tasks †

Mahshid Mehrabi [1], Shiwei Shen [1], Yilun Hai [1], Vincent Latzko [1], George P. Koudouridis [2], Xavier Gelabert [2], Martin Reisslein [3,*] and Frank H. P. Fitzek [1]

[1] Deutsche Telekom Chair of Communication Networks, Technische Universität Dresden, 01069 Dresden, Germany; mahshid.mehrabi@tu-dresden.de (M.M.); shiwei.shen@tu-dresden.de (S.S.); yilun.hai@mailbox.tu-dresden.de (Y.H.); vincent.latzko@tu-dresden.de (V.L.); frank.fitzek@tu-dresden.de (F.H.P.F.)

[2] Huawei Technologies Sweden AB, Skalholtsgatan 9, 164 40 Kista, Sweden; george.koudouridis@huawei.com (G.P.K.); xavier.gelabert@huawei.com (X.G.)

[3] School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287-5706, USA

* Correspondence: reisslein@asu.edu

† This paper is an extended version of our paper published in the Proceedings of the IEEE Globecom 2020 conference.

Citation: Mehrabi, M.; Shen, S.; Hai, Y.; Latzko, V.; Koudouridis, G.P.; Gelabert, X.; Reisslein, M.; Fitzek, F.H.P. Mobility- and Energy-Aware Cooperative Edge Offloading for Dependent Computation Tasks. *Network* **2021**, *1*, 191–214. https://doi.org/10.3390/network1020012

Academic Editor: Amitava Datta

Received: 20 July 2021
Accepted: 1 September 2021
Published: 4 September 2021

Abstract: Cooperative edge offloading to nearby end devices via Device-to-Device (D2D) links in edge networks with sliced computing resources has mainly been studied for end devices (helper nodes) that are stationary (or follow predetermined mobility paths) and for independent computation tasks. However, end devices are often mobile, and a given application request commonly requires a set of dependent computation tasks. We formulate a novel model for the cooperative edge offloading of dependent computation tasks to mobile helper nodes. We model the task dependencies with a general task dependency graph. Our model employs the state-of-the-art deep-learning-based PECNet mobility model and offloads a task only when the sojourn time in the coverage area of a helper node or Multi-access Edge Computing (MEC) server is sufficiently long. We formulate the minimization problem for the consumed battery energy for task execution, task data transmission, and waiting for offloaded task results on end devices. We convert the resulting non-convex mixed integer nonlinear programming problem into an equivalent quadratically constrained quadratic programming (QCQP) problem, which we solve via a novel Energy-Efficient Task Offloading (EETO) algorithm. The numerical evaluations indicate that the EETO approach consistently reduces the battery energy consumption across a wide range of task complexities and task completion deadlines and can thus extend the battery lifetimes of mobile devices operating with sliced edge computing resources.

Keywords: battery energy; device-enhanced edge computing; device-to-device (D2D) communication; mobility; multi-access edge computing (MEC); sliced edge computing; task dependencies; task offloading

1. Introduction

1.1. Motivation

The introduction of Multi-access Edge Computing (MEC) has facilitated the rapid growth of low-latency services provided by emerging paradigms, such as the Tactile Internet [1], the Internet of Things (IoT) [2–4], and Machine-Type-Communications (MTC) [5], as well as demanding applications, such as online gaming, virtual or augmented reality [6,7], and real-time data analytics [8]. The MEC concept brings the computation and storage resources as close as possible to the mobile end devices, namely towards the edge of the network, e.g., to cellular base stations (BSs) and WiFi access points [9–13]. Essentially, the MEC concept is a part of the ongoing trend to jointly provide communication, computing, and storage services in edge networks. In order to provide these edge network services in an economical manner, the slicing of the edge network communication, computing, and storage resources and the efficient management of the sliced resources are critical [14,15]. This study focuses on the efficient resource management of sliced computing resources in edge networks.

More specifically, this study considers sliced computing resources that are provided by installed MEC infrastructures as well as sliced computing resources that are provided by the mobile end devices with spare computing resources that are in the vicinity of a mobile device with a set of demanding computing tasks. In particular, the computing and communication resources of the adjacent end devices can be utilized to enhance the total system performance and spectral efficiency [16]. Thus, the computation tasks of a given mobile end device can be offloaded to computing slices of adjacent mobile user end devices and installed stationary MEC servers to achieve the so-called cooperative edge computing (cooperative task computing) paradigm, which is also referred to as the device-enhanced MEC paradigm [17,18].

Energy efficiency has recently emerged as an important concern for computing service provisioning in edge networks (see, e.g., [19–23]). The present study seeks to minimize the battery energy consumption on wireless end devices that operate within the device-enhanced MEC paradigm. Specifically, we consider the battery energy consumption for sets of dependent computing tasks, wherein the execution of a particular computing task may depend on the result of a preceding computing task. As elaborated in the review of related work in Section 2, the energy consumption minimization has not been previously studied in detail for mobile nodes that offload dependent computation tasks to mobile helper nodes.

1.2. Overview of Contributions and Structure of This Article

Our main contribution of this article, which extends the conference paper [24], is the development and evaluation of the online Energy-Efficient Task Offloading (EETO) algorithm that minimizes the battery energy consumption in mobile end devices with dependent computation tasks. Towards the development of EETO, we explicitly model the dependencies of the individual computation tasks of an application request in a task dependency graph in Section 3.2. We incorporate the recently developed and validated deep-learning-based PECNet user trajectory prediction into our model in Section 3.5 to account for the sojourn times of a mobile UE that offloads tasks within the coverage areas of the various prospective presently nearby (but mobile) helper nodes that can assist by taking over computation tasks or relaying computation tasks to a stationary MEC server.

We formulate the energy minimization problem in Section 4, considering the battery energy expenditures for task computation, task data transmission, and waiting for offloaded tasks to complete. In order to efficiently solve the energy minimization problem, which is a non-convex mixed integer nonlinear program, we conduct a transformation to a quadratically constrained quadratic programming (QCQP) problem in Section 5.1 and specify the EETO algorithm in Section 5.2. To the best of our knowledge, the EETO algorithm is the first task offloading algorithm in a device-enhanced MEC setting that accommodates dependent computation tasks and employs deep-learning-based trajectory prediction for both the mobile UE that offloads tasks as well as the mobile helper nodes. The performance evaluation in Section 6 indicates that EETO flexibly accommodates a wide range of scenarios and parameter settings and consistently achieves low battery energy consumption, whereas benchmarks typically perform well only for a specific scenario or narrow parameter range.

2. Related Work

2.1. General Cooperative Edge Computing Approaches

Various cooperative task computing approaches that utilize D2D task offloading to nearby end devices have recently been proposed in order to improve the MEC system performance in terms of energy efficiency [25–28], time delay [29,30], joint optimization of energy consumption and latency [31–33], IoT service enhancement [34], resource management [35–42], as well as security [43,44] and privacy [45].

2.2. Mobility Models

User mobility, i.e., the movement of devices or vehicles in the area of an MEC network, introduces several challenges for task offloading in cooperative MEC networks, which have been considered in relatively few prior studies. A matching-based algorithm for choosing the proper task offloading method to road side units (RSUs) and nearby vehicles in vehicular edge computing has been proposed in [46]. The matching-based algorithm aims at optimizing the system utility in terms of latency as well as computing and communication costs. The vehicle locations in [46] are modeled by 2D Euclidean coordinates, while their mobility follows a constant velocity model which is predetermined along roadways; a similar roadways-based mobility model has been considered in [47] (various other studies, e.g., [48], have examined similar mobility models for offloading to RSUs only, not to other vehicles). The mobility models based on roadways are appropriate for the specific case of vehicular networks but are not suitable for general mobility scenarios that are not restricted to predetermined roadways. The study [49] considered cooperative task offloading with mobile MEC servers that are mounted on unmanned aerial vehicles (UAVs), and a non-cooperative version of this UAV-assisted MEC is studied in [50]. The flight path of the UAVs is optimized to effectively support the offloading of independent tasks that can be partitioned. In contrast to these preceding studies, we consider general mobility scenarios for the end devices and stationary MEC servers.

The study [51] used a deep-learning-based algorithm to predict the user mobility trajectories and locations in general mobility scenarios to develop an online algorithm for the non-cooperative offloading of tasks from a mobile end device to stationary MEC servers. However, the study [51] did not consider the cooperation between adjacent end devices. In contrast, we consider cooperative task offloading to adjacent mobile end devices reached with D2D communication and to stationary MEC servers with cellular communication.

The study [52] considered an elementary position and direction vector mobility model for cooperative task offloading. A generic three-layer cooperative edge computing network architecture is presented in [53] taking into account the mobility effects of the users by considering the sojourn time with exponential distribution for the coverage of fog nodes. The study [53] parameterizes the exponential distribution for the sojourn times via a Gaussian distribution and notes that a future work direction is to employ machine learning for mobility modeling. The study [54] considered the cooperative task offloading of independent tasks utilizing human pedestrian trajectories that are predicted via data mining techniques [55,56]. In the present paper, we tackle the future work direction noted in [53] and advance the mobility modeling in [54] by formulating the task offloading model with the PECNet deep learning trajectory method, as elaborated in Section 3.5, in the context of dependent computation tasks.

2.3. Task Dependency Models

The existing task offloading studies typically considered independent computation tasks. To the best of our knowledge, only the non-cooperative MEC task offloading study [51] considered task dependencies. However, the task dependencies in [51] are limited to a simple sequential (linear) task dependency, where each task depends only on the immediately preceding task in a linear task sequence. A general task dependency graph has been considered in the non-cooperative task offloading study [57]. However, both studies [51,57] offloaded tasks in a non-cooperative fashion, i.e., only to installed stationary MEC and cloud servers (not to mobile helper end devices). In contrast, we consider arbitrary task dependencies that are represented by a general task dependency graph for cooperative task offloading to mobile helper end devices and stationary MEC servers.

To the best of our knowledge, the present study is the first to develop and evaluate a task offloading algorithm that minimizes the battery energy consumption for arbitrary task dependencies in a cooperative edge computing setting with mobile helper nodes. The proposed EETO algorithm employs a state-of-the-art deep-learning-based trajectory prediction for general mobility settings.

3. System Model

3.1. Overview

We consider a three-layer heterogeneous network with multiple devices and small cells. As illustrated in Figure 1, each small cell has a base station (BS), which could, for instance, be a Wi-Fi access point or a femto cell base station. In addition, each BS is affiliated with an MEC server, e.g., an MEC server could be directly attached to the BS, or the MEC service could be provided to the BS via an edge cloud network architecture [58]. The devices, which in the 3GPP standardization language are referred to as user equipment nodes, are mobile.

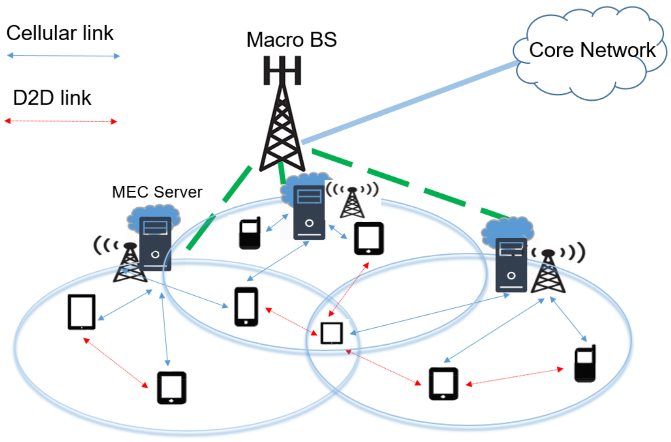


Figure 1. System model: A user equipment node (UE) with a set of computation tasks moves through the coverage regions of different base stations (BSs) with associated MEC servers, with potentially overlapping coverage areas, and within the D2D link ranges of potential helper/relay (UH) nodes. The macro BS coordinates the task offloading.

While the 3GPP standardization language uses the abbreviation UE for all user equipment nodes, we reserve the UE abbreviation for a device with a set of computation-intensive tasks. The remaining devices that have sufficient computation/communication resources to be able to act as helpers or relays, such as mobile phones, tablets, and laptop computers, are referred to as user equipment helper nodes (UHs). For simplicity of the system model, we consider a single UE with a set of computation-intensive tasks in this study. The system model with a single UE is directly applicable for scenarios that combine a low density of end devices (UEs) with computation-intensive tasks with a generally high density of UHs, such that each UE can essentially form its own surrounding cloud of adjacent D2D-connected UHs. Similar to our system model, the preceding non-cooperative dependent-task offloading studies considered either a single UE with a set of dependent tasks [51] or a set of UEs (each with one task) that, in the aggregate, form one set of dependent tasks [57]. The system model in this article can serve as a basis for future model extensions to multiple UEs (each with a set of dependent tasks) or, effectively, to multiple independent sets of dependent tasks. One possible strategy for this extension is to model the UHs with already assigned tasks as being outside of the D2D ranges of UEs that seek to offload tasks.

The UHs are randomly distributed along the UE's path, and we let $\mathcal{I} = \{\text{UH}_1, \text{UH}_2, \dots, \text{UH}_I\}$ represent the set of UHs. A $\text{UH}_i \in \mathcal{I}$ has a service coverage radius R_i. Similarly, we denote $\mathcal{M} = \{S_1, S_2, \dots, S_M\}$ as the set of edge servers, whereby server $S_m \in \mathcal{M}$ has the service coverage radius R_m. The offloading process is controlled by the macro BS (MBS), which knows the channel states and the user positions, whereby the computational load and reliability aspects of the MBS control can be addressed through decentralized control plane techniques [59]. The MBS is responsible for making the offloading decisions. The system model notations are summarized in Table 1.

Table 1. Model notations, with default parameters for evaluation in Section 6.

Symbol	Definition
$\mathcal{I}$	Set of helper/relay nodes UH$_i$, $i \in \mathcal{I}$; total # of helpers/relays $I = \lvert\mathcal{I}\rvert = 10$
$\mathcal{M}$	Set of MEC servers S_m, $m \in \mathcal{M}$; total # of MEC servers $M = \lvert\mathcal{M}\rvert = 2$
$\mathcal{K}$	Set of computation tasks; total # of tasks $K = \lvert\mathcal{K}\rvert = 25$
b_k	Data size [in bits] of task k; $b_k =$ 200–400 KB (unif. random)
c_k	Required computation per bit of task k; $c_k =$ 30–50 cycles/bit (unif. random)
$T_d^{\max}$	Deadline for execution of set of K tasks; $T_d^{\max} = 4.4$ s
B	Bandwidth of wireless channel; $B = 5$ MHz
H_k	Wireless channel gain; $H_k = 10^{-7}$ for UE to UH$_i$, and for UH$_i$ to server; $H_k = 10^{-8}$ for UE to server
P_k^{tr}	Transmission power for task k; $P_k^{tr} = 0.2$ W
P_k^{wait}	UE's idle circuit power; $P_k^{\text{wait}} = 0.05$ W
f_k^{UE}	UE CPU cycle frequency allocated to task k; $f_k^{\text{UE}} = 0.1 \cdot 10^9$ cycles/s
$f_k^{\text{UH}_i}$	CPU cycle frequency of helper node i; $f_k^{\text{UH}_i} = 0.5 \cdot 10^9$ cycles/s
$f_k^{S_m}$	CPU cycle frequency of MEC server m; $f_k^{S_m} = 2 \cdot 10^9$ cycles/s
$\mathcal{H}$	Set of helper node selection variables
$\mathcal{S}$	Set of MEC server selection variables
$\mathcal{HS}$	Set of relays and MEC server selection variables
X_t^{UE}, $X_t^{\text{UH}_i}$	Position of UE, helper/relay UH$_i$ at time t
R_i, R_m	Coverage area radius of helper UH$_i$, server S_m
$T_{s,t}^{h_i}$	UE sojourn time in coverage of helper node UH$_i$
L	# of iterations of stochastic mapping in EETO alg.; $L = 1000$

3.2. *Task Model*

We assume that the computation-intensive applications can be divided into tasks of different sizes which have to be executed within limited time frame constraints, whereby parallel execution of tasks is possible, subject to the task dependency constraints. For example, a video navigation application running on a smartphone [60] can be modeled as a set of tasks with dependencies as characterized by a general dependency graph. More formally, we denote $\mathcal{K} = \{1, 2, \ldots, K\}$ for the set of tasks in a computation-intensive application. Each task is associated with a set of parameters $\{b_k, c_k\}$, whereby b_k is the data size of computation task k (in bits), and c_k denotes the required computation resources to execute the computations for each bit in task k (in CPU cycles/bit), corresponding to a total amount of $b_k c_k$ computation resources (in CPU cycles) required to execute the task k. The computationally intensive application has a deadline $T_d^{\max}$ for the execution of all K tasks.

The dependency relationship of tasks implies an execution order, according to which a given task may have to wait for its predecessor to be executed first (see Figure 2). We define the concepts of the start and finish time of a task to model this effect for the computation offloading decision algorithm as follows:

- *Finish time* is the time instant of the execution completion of task k:

$$FT_k = ST_k + T_k^{\text{exe}}, \tag{1}$$

  whereby ST_k is the start time of task k as defined next, and T_k^{exe} is the execution time (span) for task k.
- *Start time* is the time instant when the execution of task k can commence:

$$ST_k = \begin{cases} \max\limits_{j \in \mathcal{P}(k)} FT_j & \text{if } \mathcal{P}(k) \neq \emptyset, \\ 0 & \text{if } \mathcal{P}(k) = \emptyset, \end{cases} \tag{2}$$

whereby the set $\mathcal{P}(k)$ contains all predecessor tasks of task k. According to Equation (2), the execution of a task k without predecessors ($\mathcal{P}(k) = \emptyset$) can start immediately, while the start time of a task k that depends on predecessor tasks ($\mathcal{P}(k) \neq \emptyset$) equals the maximum finish time FT_j of the respective predecessor tasks $j \in \mathcal{P}(k)$.

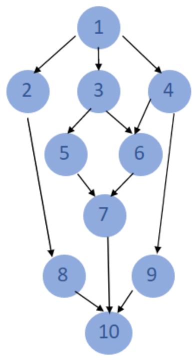


Figure 2. Example illustration of general task dependency graph specifying the required task execution order for an application with a total of $K = 10$ tasks. Tasks $k = 2$, 3, and 4 depend on the prior completion of task $k = 1$. The last task $K = 10$ has to be completed by the deadline $T_d^{\max}$.

3.3. Communication Model

The achievable up-link data rate for the transmission of task k can be obtained based on the Shannon theorem as

$$r_k = B \log_2 \left(1 + \frac{P_k^{tr} H_k}{\sigma^2} \right), \tag{3}$$

whereby B is the channel bandwidth between the sender and receiver. The sender options include the UE and the UHs, and the receivers can be the UHs and the MEC servers. We denote P_k^{tr} for the transmission power for task k, denote H_k for the channel gain between the sender and receiver while transmitting task k, and denote σ^2 for the Gaussian channel noise variance (with default value $\sigma^2 = 10^{-9}$ W). We note that more complex channel models, e.g., models that include fading and shadowing, can be substituted into our overall task offloading model in a straightforward manner and are left as a future research direction.

In our scenario, the computed results of a task are of negligible size, and their downlink transmission is not explicitly modeled. We note that in order to mitigate interference, the UE should be actively uploading (transmitting) only one task at a time to a server via the cellular channel and one task at a time to a helper or relay node via the D2D channel, whereby both cellular and D2D transmission can occur simultaneously as they do not interfere with each other. We model both the cellular channel and the D2D channel to have bandwidth B. We do not consider the transmission sequencing to at most one cellular and one D2D transmission at a time in the current model and leave this transmission sequencing as a future model refinement.

3.4. Computation Model

There are four alternative decisions for task execution in our dynamic offloading decision algorithm: (a) local execution on the UE, (b) remote execution on a helper UH, or (c) remote execution on the MEC server, whereby a task can be transmitted directly by the UE to the MEC server or via a relay (UH). We proceed to define each possible decision in detail.

3.4.1. Local Execution

If our optimization algorithm decides to execute task k locally at the UE, then the execution time is

$$T_k^l = \frac{b_k c_k}{f_k^{\mathrm{UE}}}, \tag{4}$$

where b_k is the data size of task k in bits, c_k denotes the required computational CPU cycles per bit for task k, and $f_k^{\mathrm{UE}}$is the UE's CPU cycle frequency allocated to execute task k. Based on Equation (4) and the effective switched capacitance e, which characterizes the chip architecture [61], the UE battery energy consumption for local execution can be estimated as

$$E_k^l = (f_k^{\mathrm{UE}})^2 e b_k c_k, \ \ \forall k \in \mathcal{K}, \tag{5}$$

whereby we set the default effective switched capacitance e values to $e^{\mathrm{UE}} = 10^{-25}$ F and $e^{\mathrm{UH}_i} = 0.8 \cdot 10^{-27}$ F in our evaluations in Section 6. We set the decision variable $x_k = 1$ if task k is executed locally; otherwise, $x_k = 0$.

3.4.2. Helper Execution

The task execution process by a helper UH$_i$ consists of two steps. First, the UE transfers task k to the helper UH$_i$ via D2D communication on the up-link, and then task k is executed by UH$_i$. As illustrated in Figure 3, the total execution delay is

$$T_k^{h_i} = \frac{b_k}{r_k^{\mathrm{UH}_i}} + \frac{b_k c_k}{f_k^{\mathrm{UH}_i}}, \tag{6}$$

whereby $r_k^{\mathrm{UH}_i}$ is the transmission rate from the UE toUH$_i$, and $f_k^{\mathrm{UH}_i}$ is the CPU cycle frequency of UH$_i$ allocated for the execution of task k. We neglect the delay for sending the result back from helper node UH$_i$ to the UE in our model due to the typically small number of output bits.

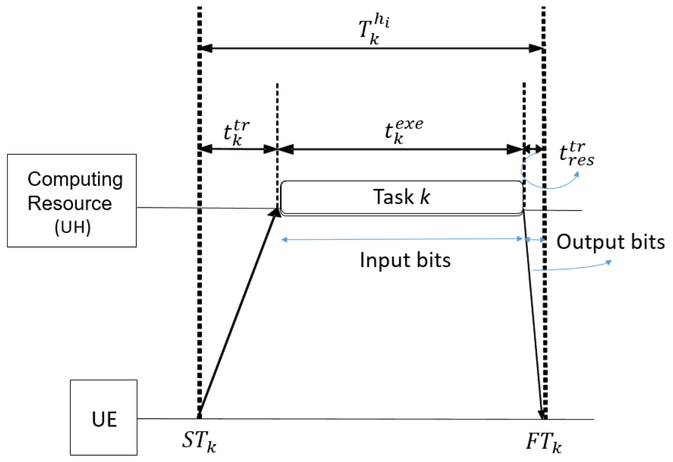


Figure 3. Timing diagram for the execution of task k by an external computing resource (helper UH$_i$): The total task execution delay $T_k^{h_i}$ consists of the transmission delay $t_k^{\mathrm{tr}} = b_k / r_k^{\mathrm{UH_i}}$, the execution delay $t_k^{\mathrm{exe}} = b_k c_k / f_k^{\mathrm{UH_i}}$, and the transmission delay for returning the computation result (which is neglected in our model).

Following Equation (6), the battery energy consumption of the helper execution mode consists of (a) the transmission from the UE to the helper UH$_i$, (b) the helper energy consumption for task execution, and (c) the energy that the UE consumes to wait for receiving the result back from the helper node:

$$E_k^{h_i} = P_k^{tr}\left(\frac{b_k}{r_k^{\mathrm{UH}_i}}\right) + e\left(f_k^{\mathrm{UH}_i}\right)^2 b_k c_k + P_k^{\mathrm{wait}}\left(\frac{b_k c_k}{f^{\mathrm{UH}_i}}\right), \tag{7}$$

where P_k^{tr} is the UE transmission power, and P_k^{wait} is the UE idle circuit power while the UE is waiting to receive the result back.

As described in Section 3.1, there are I number of helpers available for the UE in the area. Let $h_k^i = 1, h_k^i \in \mathcal{H}$, indicate that the UE chose to offload task k to UH$_i$, whereby the set $\mathcal{H} = \{h_k^i | k \in \mathcal{K}, i \in \mathcal{I}\}$ contains all helper node selection variables. Since task k can only be offloaded to at most one helper at the same time, an offloading selection algorithm should follow the constraint

$$\sum_{i \in \mathcal{H}} h_k^i \leq 1 \;\; \forall k \in \mathcal{K}. \tag{8}$$

3.4.3. Server Execution

For task execution on the server, there are two possible paths for offloading the tasks.

3.4.3.1. Direct Offloading from UE to MEC Server

The time delay for executing task k can be calculated as

$$T_k^{s_m} = \frac{b_k}{r_k^{S_m}} + \frac{b_k c_k}{f_k^{S_m}}, \tag{9}$$

where $r_k^{S_m}$ is the transmission rate from the UE to server S_m, and $f_k^{S_m}$ denotes the CPU computation cycle frequency of server S_m for the execution of task k. The corresponding UE battery energy consumption is

$$E_k^{s_m} = P_k^{tr}\left(\frac{b_k}{r_k^{S_m}}\right) + P_k^{\mathrm{wait}}\left(\frac{b_k c_k}{f_k^{S_m}}\right), \tag{10}$$

In Equation (10), the MEC server energy consumption for executing task k is not included, since MEC servers do not typically rely on battery power. Throughout this study, the focus is on saving battery energy consumption. Incorporating the saving of energy consumption in the MEC servers, which are powered from the wired grid, is an interesting direction for future research.

Since there are M servers available for the UE in the area, the UE can choose to offload task k to S_m, and in this case, $s_k^m = 1$, whereby the set $s_k^m \in \mathcal{S}$, $\mathcal{S} = \{s_k^m | k \in \mathcal{K}, m \in \mathcal{M}\}$ contains all MEC server node selection variables. Since a given task k can only be offloaded to one MEC server (and not split among multiple MEC servers), the offloading selection algorithm should follow the constraint

$$\sum_{m \in \mathcal{M}} s_k^m \leq 1 \;\; \forall k \in \mathcal{K}. \tag{11}$$

3.4.3.2. Offloading from UE via Relay UH_i to MEC Server

The UE first sends task k to the relay UH$_i$, which then forwards the task to the server S_m. The delay consists of three steps for transmissions and computation:

$$T_k^{s_{i,m}} = \frac{b_k}{r_k^{\mathrm{UH}_i}} + \frac{b_k}{r_k^{S_{m,i}}} + \frac{b_k c_k}{f_k^{S_m}}, \tag{12}$$

whereby $r_k^{\mathrm{UH}_i}$ is the transmission rate from the UE to UH$_i$, and $r_k^{S_{m,i}}$ is the transmission rate from UH$_i$ to server S_m. The UE and UH battery energy consumption is

$$E_k^{s_{i,m}} = P_k^{tr}\left(\frac{b_k}{r_k^{\mathrm{UH}_i}}\right) + P_k^{\mathrm{wait}}\left(\frac{b_k}{r_k^{S_{m,i}}} + \frac{b_k c_k}{f_k^{S_m}}\right) + P_k^{tr,i}\left(\frac{b_k}{r_k^{S_m}}\right), \tag{13}$$

where $P_k^{tr,i}$ is the transmission power of UH_i. We assume that the computation result is sent back directly from server S_m to the UE and that the transmission delay for the result is negligible. We define the selection variables $s_k^{i,m} = 1, s_k^{i,m} \in \mathcal{HS}$ for offloading task k via UH_i to S_m, whereby the set $\mathcal{HS} = \{s_k^{i,m} | k \in \mathcal{K}, i \in \mathcal{I}, m \in \mathcal{M}\}$ contains all relays and MEC server selection variables. Since a given task k can only be serviced via a single helper node UH_i by a single server S_m, the offloading selection algorithm should follow the constraint

$$\sum_{m\in\mathcal{M}} \sum_{i\in\mathcal{I}} s_k^{i,m} \le 1 \;\; \forall k \in \mathcal{K}. \tag{14}$$

3.5. Mobility Model

Based on recent studies, most user trajectories contain similar patterns [51,62]. To incorporate this insight so as to achieve effective mobility-aware task offloading, we employ machine learning to predict the UE's and the UHs' paths to estimate their available service coverage time. Specifically, we employ a Predicted Endpoint Conditioned Network (PECNet) [63] as a deep-learning-based method for predicting socially compliant trajectories which infer users' destinations to assist prediction. This enhances the plausibility of predictions for trajectories in addition to using the historical data of motion paths, yielding coherent user trajectories. The main idea of PECNet is to divide the prediction problem into two parts. The first part estimates the potential destinations of users using an endpoint estimation variational autoencoder (VAE). The second part predicts socially compliant trajectories while jointly considering the motion history and potential destinations of all users in the scene.

The PECNet system model [63], which is illustrated in Figure 4, includes three key elements: a past trajectory encoder, endpoint VAE, and social pooling module. First, a user's motion histories are encoded via the past trajectory encoder. Then, the result is fed into the endpoint VAE to estimate the user's destination. Subsequently, the social pooling module uses the encoded past trajectory and the estimated destinations of all users to jointly predict the future path of every user in the scene. The final output are paths whose future segments (i.e., the predictions) are strongly dependent on the past locations (i.e., the inputs).

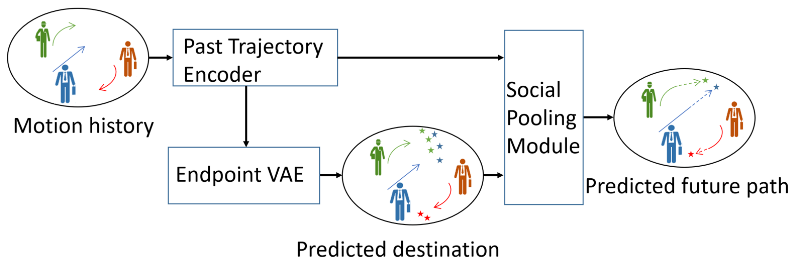


Figure 4. PECNet system model [63]: Past trajectory encoder and endpoint estimation variational encoder (VAE) feed into social pooling module to predict paths.

Similar to prior studies involving mobility prediction, e.g., [12,54], we consider time to be suitably slotted (whereby we consider the typical 0.4 s slot duration for pedestrian mobility). In our scenario, the positions of the UE and a helper node UH_i at time t are defined as $X_t^{\mathrm{UE}} = (x^t, y^t)$ and $X_t^{\mathrm{UH}_i} = (x_i^t, y_i^t)$, respectively. The trajectories of the UE and helpers UH_i from time $(t-n+1)$ to t are:

$$\mathbf{X}^{\mathrm{UE}} = \left\{X_{t-n+1}^{\mathrm{UE}}, X_{t-n+2}^{\mathrm{UE}}, \ldots, X_t^{\mathrm{UE}}\right\} \tag{15}$$

$$\mathbf{X}^{\mathrm{UH}_i} = \left\{X_{t-n+1}^{\mathrm{UH}_i}, X_{t-n+2}^{\mathrm{UH}_i}, \dots, X_t^{\mathrm{UH}_i}\right\}. \tag{16}$$

This information can be collected by the macro BS. The PECNet takes as input the trajectories of the UE and helpers UH$_i$ and outputs the predicted movements from time $(t+1)$ to $(t+l)$:

$$\widehat{\mathbf{Y}}^{\mathrm{UE}} = \left\{\widehat{Y}_{t+1}^{\mathrm{UE}}, \widehat{Y}_{t+2}^{\mathrm{UE}}, \dots, \widehat{Y}_{t+l}^{\mathrm{UE}}\right\} \tag{17}$$

$$\widehat{\mathbf{Y}}^{\mathrm{UH}_i} = \left\{\widehat{Y}_{t+1}^{\mathrm{UH}_i}, \widehat{Y}_{t+2}^{\mathrm{UH}_i}, \dots, \widehat{Y}_{t+l}^{\mathrm{UH}_i}\right\}, \tag{18}$$

where $\widehat{Y}_t^{\mathrm{UE}} = (\hat{x}^t, \hat{y}^t)$ and $\widehat{Y}_t^{\mathrm{UH}_i} = (\hat{x}_i^t, \hat{y}_i^t)$. A D2D link can be established between the UE and helper UH$_i$ at time t if the UE is in the coverage area of helper UH$_i$ with radius R_i, i.e., if

$$\sqrt{(\hat{x}^t - \hat{x}_i^t)^2 + (\hat{y}^t - \hat{y}_i^t)^2} \leq R_i. \tag{19}$$

The sojourn time of the UE in UH$_i$'s coverage from time t is then

$$T_{s,t}^{h_i} = \max\ l, \text{ s.t. } \sqrt{(\hat{x}^\tau - \hat{x}_i^\tau)^2 + (\hat{y}^\tau - \hat{y}_i^\tau)^2} \leq R_i, \ \forall \tau \in [t, t+l], \tau \in \mathbb{Z}. \tag{20}$$

The same process can be utilized to obtain the sojourn time $T_{s,t}^{s_m}$ of the UE in the coverage of MEC server S_m, as well as the sojourn time $T_{s,t}^{s_{i,m}}$ of UH$_i$ in the coverage of MEC server S_m.

As we observe from Figure 1, due to the users' mobility, the service coverage is limited; however, in our method, by defining the concept of sojourn time $T_{s,t}$ and finish time FT_k of the tasks, users only choose a destination if the period of the coverage availability ($T_{s,t}$) is longer than the required time (FT_k) for the task execution.

The validations in [63] indicate that PECNet is highly accurate. We assume that the PECNet predictions are correct, similar to related studies that assume perfectly correct mobility predictions, e.g., [48,53]. Various mechanisms for mitigating any rarely occurring prediction errors can be examined in future research. One strategy for reducing the chance of prediction errors could be to impose "safety margins" on the predicted sojourn times. Another strategy could be to classify the tasks according to their level of criticality and to classify the helpers according to their reliability; then, tasks could be assigned under consideration of the task criticality and helper reliability to minimize potential offloading failures.

4. Dynamic Computation Offloading Problem Formulation

Considering the offloading decision options defined in Section 3, our joint cost optimization of the computation and communication cooperation in the device-enhanced MEC system for the execution of task k considering the task's execution deadline can be defined based on the battery energy consumption for the execution of task k:

$$E_k^{\mathrm{exe}} = x_k E_k^l + h_k^i E_k^{h_i} + s_k^m E_k^{s_m} + s_k^{i,m} E_k^{s_{i,m}}, \tag{21}$$

where x_k, h_k^i, s_k^m, and $s_k^{i,m}$ represent the binary variables for the offloading decision according to the strategies introduced in Section 3.4. For the execution of task k, only one of these variables can be 1, and the rest are 0, giving the task execution time

$$T_k^{\mathrm{exe}} = x_k T_k^l + h_k^i T_k^{h_i} + s_k^m T_k^{s_m} + s_k^{i,m} T_k^{s_{i,m}}. \tag{22}$$

Hence, the finish time FT_k of task k is obtained by inserting T_k^{exe} from Equation (22) into Equation (1).

The optimization problem of minimizing the energy cost considering the service execution deadline, the mobility of the end devices, and the task dependencies can be formulated based on the total energy consumption $E_{\text{tot}}^{\text{exe}} = \sum_{k \in \mathcal{K}} E_k^{\text{exe}}$:

$$
\begin{aligned}
OP1: \quad & \min_{\alpha,\beta} \quad E_{\text{tot}}^{\text{exe}} \\
s.t: C1: \; & x_k, h_k^i, s_k^m, s_k^{i,m} \in \{0,1\}, \; \forall k \in \mathcal{K}, \; \forall i \in \mathcal{I}, \; \forall m \in \mathcal{M} \\
C2: \; & x_k + \sum_{i=1}^{I} h_k^i + \sum_{m=1}^{M} s_k^m + \sum_{m=1}^{M} \sum_{i=1}^{I} s_k^{i,m} = 1, \; \forall k \in \mathcal{K} \\
C3: \; & ST_k = \begin{cases} \max\limits_{j \in \mathcal{P}(k)} FT_j & \text{if } \mathcal{P}(k) \neq \emptyset, \\ 0 & \text{if } \mathcal{P}(k) = \emptyset, \end{cases} \quad \forall k \in \mathcal{K} \\
C4: \; & FT_K \leq T_d^{\max}, \\
C5: \; & FT_k \leq T_{s,k}, \quad \forall k \in \mathcal{K},
\end{aligned}
\tag{23}
$$

where $\alpha = [x_1, h_1^1, \ldots, h_1^I, s_1^1, \ldots, s_1^M, s_1^{1,1}, \ldots, s_1^{I,M}, \ldots, x_K, h_K^1, \ldots, h_K^I, s_K^1, \ldots, s_K^M, s_K^{1,1}, \ldots, s_K^{I,M}]$ are the per-task offloading decision making variables, and $\beta = [ST_1, ST_2, \ldots, ST_K]$ are the start times. The decision-making variables are present in the constraints C1 and C2. Constraint C3 defines the start time for each task k based on its predecessor tasks. The execution of tasks without predecessor tasks can start immediately, and the start time of all other tasks equals the maximum finish time of their respective predecessors. Constraint C4 indicates that the finish time of the last task K should be less than or equal to the total time deadline $T_d^{\max}$ of the application.

Constraint C5 represents that an offloading destination may only be chosen if the sojourn time $T_{s,k}$ of the UE in the range of the computation resource is longer than the time FT_k needed to finish executing task k. Utilizing Equation (20), the sojourn time for the different offloading options can be calculated as

$$
T_{s,k} = \begin{cases} T_{s,t}^{h_i} & \text{if} \quad h_k^i = 1 \\ T_{s,t}^{s_m} & \text{if} \quad s_k^m = 1 \\ \min\{T_{s,t}^{h_i}, T_{s,t}^{s_m}, T_{s,t}^{s_{i,m}}\} & \text{if} \quad s_k^{i,m} = 1. \end{cases}
\tag{24}
$$

In Equation (24), $T_{s,t}^{h_i}$ denotes the UE sojourn time in the range of helper node UH$_i$, $T_{s,t}^{s_m}$ is the UE sojourn time in the range of MEC server S_m, and $T_{s,t}^{s_{i,m}}$ is the sojourn time of helper UH$_i$ in the coverage area of MEC server S_m. If the decision is made to offload task k to either the helper node UH$_i$ ($h_k^i = 1$) or MEC server S_m ($s_k^m = 1$), the finish time FT_k of task k should be shorter than the sojourn time $T_{s,k}$ in the coverage of that computation resource, meaning that any chosen resource should remain in range until the completion of the execution of task k. The offloading to server S_m via relaying by helper node UH$_i$ requires that the UE is in the coverage area of the helper (with corresponding sojourn time $T_{s,t}^{h_i}$), that the helper is in the coverage of the server ($T_{s,t}^{s_{i,m}}$), and that the UE is still in the coverage of the server ($T_{s,t}^{s_m}$) for directly returning the computation results from the server to the UE.

We further note that the sojourn time $T_{s,k}$ in constraint C5 depends on the time t, as explicitly indicated by the t subscript on the right-hand side of Equation (24) and in Equation (20). In order to avoid notational clutter, we have omitted the t subscript on the left-hand side of Equation (24) and in the optimization problem in Equation (23). However, we emphasize that the optimization problem in Equation (23) is solved in an online fashion at a particular time t when a UE request for computing a set of $\mathcal{K}$ tasks arrives. The start time ST_k of a task (with $ST_1 = 0$) is relative to the UE request arrival time t.

5. Solution of Dynamic Computation Offloading Optimization Problem

The optimization problem $OP1$ is a non-convex mixed integer nonlinear programming problem due to the binary constraints and, therefore, hard to solve in polynomial time [64]. To facilitate an efficient solution, it should first be transformed to an equivalent quadratically constrained quadratic programming (QCQP) problem. Then, the binary offloading decisions can be recovered using a semidefinite relaxation (SDR) and stochastic mapping method. The QCQP can be efficiently solved, e.g., with conic optimization solvers [65], especially with accelerators [66,67]. On a contemporary personal computer with a moderate performance level, the QCQP solution takes on the order of 1 second for the typical problems considered in the evaluation in Section 6. With the appropriate computing hardware at BSs and suitable accelerators, solution times on the order of 0.1 seconds or less are anticipated, making the online solution of the optimization feasible for scenarios with pedestrian-level mobility. We outline additional strategies for speeding up the offloading decision making in future research in Section 7.2.

5.1. Conversion of $OP1$ into QCQP

The first conversion step for OP1 is to convert the integer constraints to a quadratic formulation:

$$\begin{aligned} &x_k(x_k-1)=0,\ h_k^i\left(h_k^i-1\right)=0,\ s_k^m(s_k^m-1)=0,\\ &s_k^{i,m}\left(s_k^{i,m}-1\right)=0,\ \forall k\in\mathcal{K},\forall i\in\mathcal{I},\forall m\in\mathcal{M}. \end{aligned} \tag{25}$$

With these quadratic constraint formulations, the QCQP transformation of OP1 becomes

$$\begin{aligned} OP2:\ &\min_{\alpha,\beta}\ E_{\text{tot}}^{\text{exe}}\\ s.t: C1:\ & x_k(x_k-1)=0,\ h_k^i\left(h_k^i-1\right)=0,\\ & s_k^m(s_k^m-1)=0,\ s_k^{i,m}\left(s_k^{i,m}-1\right)=0,\\ & \forall k\in\mathcal{K},\forall i\in\mathcal{I},\forall m\in\mathcal{M};\\ & C2: x_k+\sum_{i=1}^{I}h_k^i+\sum_{m=1}^{M}s_k^m+\sum_{m=1}^{M}\sum_{i=1}^{I}s_k^{i,m}=1,\ \forall k\in\mathcal{K};\\ & C3_0: ST_k=0\ \ \forall j\in\mathcal{P}(k),\mathcal{P}(k)=\emptyset,\ \forall k\in\mathcal{K};\\ & C3_1: ST_k-FT_j\geq 0,\ \ \forall j\in\mathcal{P}(k),\mathcal{P}(k)\neq\emptyset,\ \ \forall k\in\mathcal{K};\\ & C4: FT_K\leq T_d^{\max};\quad C5: FT_k\leq T_{s,k},\ \forall k\in\mathcal{K}; \end{aligned} \tag{26}$$

where we reformulated $C1$ and vectorized C3.

Next, we define a vector v of dimension $((2+I+M+IM)K+1)\times 1$ as $v=[\alpha,\beta,1]^T$, and a standard unit vector e_j with the jth entry equal to 1 and dimension $((2+I+M+IM)K+1)\times 1$. Furthermore, we define

$$\begin{aligned} v=&[x_1,h_1^1,\ldots,h_1^I,s_1^1,\ldots,s_1^M,s_1^{1,1},\ldots,s_1^{I,M},\ldots,x_K,\\ &h_K^1,\ldots,h_K^I,s_K^1,\ldots,s_K^M,s_K^{1,1},\ldots,s_K^{I,M},ST_1,\ldots,ST_K,1]^T,\\ n_0=&[E_1^l,E_1^{h_1},\ldots,E_1^{h_I},E_1^{s_1},\ldots,E_1^{s_M},E_1^{s_{1,1}},\ldots,\\ &E_1^{s_{I,M}},\ldots,E_K^l,E_K^{h_1},\ldots,E_K^{h_I},E_K^{s_1},\ldots,E_K^{s_M},\\ &E_K^{s_{1,1}},\ldots,E_K^{s_{I,M}},\mathbf{0}_{1\times(K+1)}] \end{aligned} \tag{27}$$

as the set of decisions to which devices the tasks should be offloaded to or through, respectively; n_0 is the vector of energy consumptions associated with a full set of decisions, extended by a row of zeros for the optimizer, and

$$\begin{aligned}
n_{1k} =& e_{(1+I+M+IM)(k-1)+1} + \cdots + e_{(1+I+M+IM)k-1} \\
&+ e_{(1+I+M+IM)k} + e_{(1+I+M+IM)K+k} \\
n_2 =& [0_{1\times(2+I+M+IM)(K-1)}, T_K^l, T_K^{h_1}, \ldots, T_K^{h_I}, \\
& T_K^{s_1}, \ldots, T_K^{s_M}, T_K^{s_{1,1}}, \ldots, T_K^{s_{I,M}}, 0_{1\times(K-1)}, 1, 0]^T \\
n_3 =& e_{(2+I+M+IM)(j-1)+1} + \cdots + e_{(2+I+M+IM)j-1} \\
&+ e_{(2+I+M+IM)j} + e_{(2+I+M+IM)K+j} \\
n_T =& [T_1^l, T_1^{h_1}, \ldots, T_1^{h_I}, T_1^{s_1}, \ldots, T_1^{s_M}, T_1^{s_{1,1}}, \ldots, \\
& T_1^{s_{I,M}}, \ldots, T_K^l, T_K^{h_1}, \ldots, T_K^{h_I}, T_K^{s_1}, \ldots, T_K^{s_M}, \\
& T_K^{s_{1,1}}, \ldots, T_K^{s_{I,M}}, 1_{1\times(K+1)}]^T \\
n_{T_s} =& [T_d^{\max}, T_{s,t}^{h_1}, \ldots, T_{s,t}^{h_I}, T_{s,t}^{s_1}, \ldots, T_{s,t}^{s_M}, T_{s,t}^{s_{1,1}}, \ldots, \\
& T_{s,t}^{s_{I,M}}, \ldots, T_d^{\max}, T_{s,t}^{h_1}, \ldots, T_{s,t}^{h_I}, T_{s,t}^{s_1}, \ldots, T_{s,t}^{s_M}, \\
& T_{s,t}^{s_{1,1}}, \ldots, T_{s,t}^{s_{I,M}}, 1_{1\times(K+1)}]^T
\end{aligned} \tag{28}$$

are the reformulated equations for the constraints, consisting of standard unit vectors as well as task execution times and sojourn times. The vector n_{1k} corresponds to the $(1+I+M+IM)$ probabilities of the offloading strategies for task k and additionally the task k deadline (used in C5). In the definition of n_2, T_K^l is the local task K execution time from Equation (4), $T_K^{h_i}$ is the UH$_i$ task K execution time from Equation (6), $T_K^{s_m}$ is the server m task K execution time from Equation (9), and $T_K^{s_{i,m}}$ is the task k execution time with relaying via UH$_i$ to server m from Equation (12); these execution times for the various tasks k, $k = 1, \ldots, K$, are also used in the definition of n_T. The vector n_{T_s} of the sojourn times has the same dimension as n_T. Thus, the QCQP transformation of $OP2$ can be written as

$$\begin{aligned}
OP3: \quad & \min_{v} \quad (n_0)^T v \\
s.t: & C1: v^T \mathrm{diag}(e_j) v - (e_j)^T v = 0, \;\; j = 1, \ldots, (1+I+M+IM)K; \\
& C2: (n_{1k})^T v = 1, \; \forall k \in \mathcal{K}; \\
& C3_0: (e_{(1+I+M+IM)K+1})^T v = 0, \;\; \forall j \in \mathcal{P}(k), \mathcal{P}(k) = \emptyset, \; \forall k \in \mathcal{K}; \\
& C3_1: (e_{(1+I+M+IM)K+k})^T v \geq (n_T)^T \mathrm{diag}(n_3) v, \;\; \forall j \in \mathcal{P}(k), \mathcal{P}(k) \neq \emptyset, \; \forall k \in \mathcal{K}; \\
& C4: (n_2)^T v \leqslant T_d^{\max}; \\
& C5: (n_T - n_{T_s})^T \mathrm{diag}(n_{1k}) v \leq 0, \;\; \forall k \in \mathcal{K}.
\end{aligned} \tag{29}$$

For the further change to the *homogeneous* QCQP $OP4$, we define $g = [v^T 1]$, $n'_1 = \mathrm{diag}(n_{1k})$, $n'_3 = \mathrm{diag}(n_3)$, $a = (2+I+M+IM)K$, $b = (1+I+M+IM)K+1$, and $c = (1+I+M+IM)K+k$. In order to convert the necessary constraints for the variety of device roles into the standard form for QCQP solvers, we convert all constraints into the matrix form

$$
\begin{aligned}
M_0 &= \begin{bmatrix} 0_{(a+1)\times(a+1)} & \frac{1}{2}n_0 \\ \frac{1}{2}(n_0)^T & 0 \end{bmatrix}; \\
M_1 &= \begin{bmatrix} \mathrm{diag}(e_j) & -\frac{1}{2}e_j \\ -\frac{1}{2}(e_j)^T & 0 \end{bmatrix}; \\
M_2 &= \begin{bmatrix} 0_{(a+1)\times(a+1)} & \frac{1}{2}n_{1k} \\ \frac{1}{2}(n_{1k})^T & 0 \end{bmatrix}; \\
M_3 &= \begin{bmatrix} 0_{(a+1)\times(a+1)} & \frac{1}{2}e_b \\ \frac{1}{2}(e_b)^T & 0 \end{bmatrix}; \\
M_3' &= \begin{bmatrix} 0_{a\times a} & -\frac{1}{2}((n_T)^T n_3')^T & \frac{1}{2}(e_c)^T \\ -\frac{1}{2}((n_T)^T n_3') & 0 & 0 \\ \frac{1}{2}(e_c)^T & 0 & 0 \end{bmatrix}; \\
M_4 &= \begin{bmatrix} 0_{(a+1)\times(a+1)} & \frac{1}{2}(n_2) \\ \frac{1}{2}(n_2)^T & 0 \end{bmatrix}; \\
M_5 &= \begin{bmatrix} 0_{(a+1)\times(a+1)} & \frac{1}{2}[(n_T - n_{T_s})^T n_{1k}']^T \\ \frac{1}{2}[(n_T - n_{T_s})^T n_{1k}'] & 0 \end{bmatrix}.
\end{aligned}
$$

Then,

$$
\begin{aligned}
OP4: \quad & \min_{g} \quad g^T(M_0)g \\
s.t: C1: & \, g^T M_1 g = 0, \; j = 1, \ldots, (1+I+M+IM)K; \\
& C2: g^T M_2 g = 1, \forall k \in \mathcal{K}, \\
& C3_0: g^T M_3 g = 0, \; \forall j \in \mathcal{P}(k), \mathcal{P}(k) = \emptyset, \; \forall k \in \mathcal{K}; \\
& C3_1: g^T M_3' g \geq 0, \; \forall j \in \mathcal{P}(k), \mathcal{P}(k) \neq \emptyset, \; \forall k \in \mathcal{K}; \\
& C4: g^T M_4 g \leqslant T_d^{\max}; \\
& C5: g^T M_5 g \leq 0, \quad \forall k \in \mathcal{K}.
\end{aligned}
\tag{30}
$$

For the final conversion step, we define the symmetric, positive semidefinite matrix $G = gg^T$ and write with $\mathrm{Tr}(\cdot)$ denoting the trace of a square matrix:

$$
\begin{aligned}
OP5: \quad & \min_{G} \quad \mathrm{Tr}(M_0 G) \\
s.t: C1: & \, \mathrm{Tr}(M_1 G) = 0, \; j = 1, \ldots, (1+I+M+IM)K; \\
& C2: \mathrm{Tr}(M_2 G) = 1, \; \forall k \in \mathcal{K}; \\
& C3_0: \mathrm{Tr}(M_3 G) = 0, \; \forall j \in \mathcal{P}(k), \mathcal{P}(k) = \emptyset, \forall k \in \mathcal{K}; \\
& C3_1: \mathrm{Tr}(M_3' G) \geq 0, \; \forall j \in \mathcal{P}(k), \mathcal{P}(k) \neq \emptyset, \; \forall k \in \mathcal{K}; \\
& C4: \mathrm{Tr}(M_4 G) \leqslant T_d^{\max}; \\
& C5: \mathrm{Tr}(M_5 G) \leq 0, \quad \forall k \in \mathcal{K}; \\
& C6: G[a+1, a+1] = 1; \\
& C7: G[a+1, a+2] = 1; \\
& C8: G[a+2, a+1] = 1; \\
& C9: G[a+2, a+2] = 1; \\
& C10: \mathrm{rank}(G) = 1.
\end{aligned}
\tag{31}
$$

While this problem $OP5$ has a few additional constraints, the single constraints are much easier to handle for the solver. For example, constraints $C6$–$C9$ are simple checks for single elements of the matrix G.

5.2. Energy-Efficient Task Offloading (EETO) Algorithm

In this section, we propose a stochastic mapping method to obtain the optimized offloading strategy according to the inter-task dependency and the whole application completion time $T_d^{\max}$. We apply SDR by dropping the last non-convex constraint of rank 1 to obtain an approximate solution $\tilde{G}$, the last row of which includes $[\alpha, \beta, 1, 1]$. If $\text{rank}(\tilde{G}) = 1$, then we directly extract α as an offloading decision for all tasks from the last row of $\tilde{G}$. Otherwise, we employ a probability-based stochastic mapping method to recover the solution. For each task k, we select the largest value of each offloading decision from the elements group with index k in α and denote them as t_1, t_2, t_3, and t_4. We map t_1, t_2, t_3, and t_4 with the probability-based stochastic mapping method:

$$q_1 t_1 + q_2 t_2 + q_3 t_3 + q_4 t_4 = 1 \tag{32}$$

$$Q_1 = q_1 t_1, Q_2 = q_2 t_2, Q_3 = q_3 t_3, Q_4 = q_4 t_4, \tag{33}$$

where Q_1, Q_2, Q_3, and Q_4 are the probabilities of the corresponding offloading decision being 1. We randomly set $t_i = 1$ according to the probabilities Q_i, and the other elements in α are set to 0. For example, when $I = 2, M = 2$, and $K = 3$, then the dimension of vector α would be $(1 + I + M + IM)K = 27$. For each task, there are $1 + I + M + IM = 9$ elements; if the solution were not rank 1, each element could be a value between 0 and 1 and the sum of the nine elements equals 1. Taking out the first nine elements in vector α, for instance, as $[0.25, 0.15, 0.05, 0.05, 0.10, 0.10, 0.05, 0.15, 0.10]$, we can find that the largest numbers of each offloading decision are 0.25, 0.15, 0.10, and 0.15, i.e., $t_1 = 0.25$, $t_2 = 0.15$, $t_3 = 0.10$, and $t_4 = 0.15$. According to Equations (32) and (33), we stochastically map the probabilities that the corresponding offloading decision would be selected with Q_1, Q_2, Q_3, and Q_4, respectively. Then, we randomly set one of the two numbers to 1, while we set the rest of the nine elements to 0, which means that only one offloading decision would be selected for the currently considered task. For each task, we perform the same stochastic mapping, and after a decision has been made for each task, we obtain the offloading strategy $\tilde{\alpha}$. Furthermore, we compare FT_K with $T_d^{\max}$, and the strategy could be a final solution only if $FT_K \leq T_d^{\max}$. For higher accuracy, we repeat the process L times to obtain a set of solutions and select the solution that yields the minimum energy cost. The algorithm can be summarized as Algorithm 1.

We note that if none of the task execution strategies from Section 3.4 are feasible for a task, then the task is too complex, and the solution of the QCQP will fail, i.e., there is no feasible solution for the EETO algorithm.

Algorithm 1: Energy-Efficient Task Offloading (EETO) Algorithm

```
Input: $L, \mathcal{K}, \mathcal{I}, \mathcal{M}, T_d^{\max}, B, \mathcal{P}(k), b_k, c_k, P_k^{tr}, P_k^{\text{wait}}, f_k^{\text{UE}}, f_k^{\text{UH}_i}, f_k^{S_m}, \forall k \in \mathcal{K}, \mathbf{X}^{\text{UE}}$,
    $\mathbf{X}^{\text{UH}_i}$ (see Table 1)
Output: Offloading strategy $\alpha$
Initialize: Predict paths of UE and helpers with PECNet with historical paths $\mathbf{X}^{\text{UE}}$,
  $\mathbf{X}^{\text{UH}_i}$, calculate sojourn times $T_{s,k}$ with Equation (24). Initialize all matrices in
  Equation (31). Solve the SDP in Equation (31) without the rank-1 constraint to get
  the optimal solution $\tilde{G}$. Extract the first $(1+I+M+IM)K$ elements of the last
  row in $\tilde{G}$ as $\alpha'$;
if rank($\tilde{G}$)==1 then
  $\alpha = \alpha'$;
else
  for $l$ = 1:$L$ do
    for $k = 1 : K$ do
      $s(k) = (1+I+M+IM)$ elements in $\alpha'(l)$ related to task $k$;
      Perform probab. based stoch. mapping: Set one element of $s(k)$ to 1,
        and others to 0;
      Compute the current $FT_k$ by $\alpha'(l)$ and Equations (1) and (22);
      if $FT_k > T_{s,k}$ then
        Discard $\alpha'(l)$;
    if $FT_K > T_d^{\max}$ then
      Discard $\alpha'(l)$;
    else
      $\alpha'(l)$ saved as a feasible solution and calculate total energy
        consumption by $\alpha'(l)$ and Equation (21);
  Select the offloading strategy $\tilde{\alpha}$ among $\alpha'(1)$,...,$\alpha'(L)$ that yields the minimum
    energy cost;
  $\alpha = \tilde{\alpha}$.
```

6. EETO Evaluation

This section presents simulation results to evaluate the performance of our joint communication and computation cooperation offloading method (EETO) for mobile helper nodes with dependent computation tasks.

6.1. Simulation Setup

The simulations employ the parameters in Table 1. The UE and UHs are initially randomly distributed and follow the mobility pattern of the publicly available PECNet dataset [63]. The coverage ranges of the BSs (servers) and each user end device are 400 m and 50 m, respectively, i.e., the UE can reach a BS that is 400 m away, but the UE can only reach a UH$_i$, $i = 1, \ldots, I$, that is 50 m or less away. We employ the PECNet model for the Stanford Drone Dataset, which is a common mobility benchmark containing over 11,000 unique pedestrians in a university campus setting [63]. Initially, we set the UE mobility speed to 1 m/s.

As outlined in Section 2.3, there is no prior benchmark for offloading dependent computation tasks in a cooperative edge computing setting. Therefore, we compare EETO with the following four alternate task offloading approaches for the dependent-task edge computing setting: all local (ALO), all sever (ASO), computation cooperation optimization (CPCO), and communication cooperation optimization (CMCO). In ALO, the tasks are all executed locally on the UE (corresponding to local execution, Section 3.4.1). In ASO, all tasks are executed remotely on MEC servers (i.e., directly offloaded from the UE to an MEC server, Section 3.4.3.1). In CPCO, the computation resources of adjacent devices can be used, and therefore, the UH$_i$ can act as helper nodes, i.e., the options from Sections 3.4.1, 3.4.2, and 3.4.3.1 are available to the optimization. In CMCO, the UH$_i$

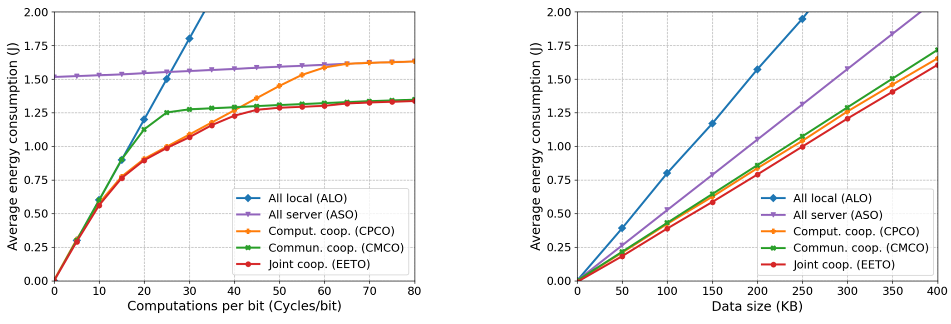
can only act as relay nodes to transmit the computation tasks to an MEC server, i.e., the options from Sections 3.4.1, 3.4.3.1, and 3.4.3.2 are available to the optimization. Thus, the main difference between CPCO and CMCO pertains to the function of the helper nodes UH$_i$: in CPCO, the UH$_i$ help only with task execution (computation) but not with relay communication, whereas in CMCO, the UH$_i$ help only with relay communication for task offloading to the MEC server but not with helper execution.

We ran 50 independent simulation replications for each evaluation. The resulting 95% confidence intervals are less than 5% of the corresponding sample means and are omitted from the plots to avoid visual clutter.

6.2. Simulation Results

6.2.1. Impact of Task Complexity c_k and Size b_k

The average battery energy consumption as a function of the computation CPU cycles c_k needed for execution of each bit of task k for the five methods is shown in Figure 5a. We observe from Figure 5a that the ALO energy consumption increases linearly as the computations c_k per bit increase. Specifically, when the required CPU cycles c_k per bit are less than 10 cycles/bit, ALO attains the minimum energy consumption. The left part of Table 2 indicates that when the required computations c_k increase to 15 cycles/bit, ALO can still satisfy the deadline requirements (marked in blue); however, the energy consumption is higher than for the other algorithms (see Figure 5a). The left part of Table 2 indicates that for computation-intensive tasks with $c_k \geq 20$ cycles/bit, ALO fails to finish the task execution within the deadline $T_d^{\max}$ (marked in red), which clearly demonstrates the need for a task offloading method.



(**a**) (**b**)

Figure 5. Average battery energy consumption $E_{\text{tot}}^{\text{exe}}$ vs. task computation demands c_k and data size b_k; default b_k, c_k, and $T_d^{\max}$ from Table 1; random task dependency. (**a**) Task computation demands c_k, (**b**) data size b_k.

Table 2. Finish time FT_K of local computing (ALO) for different computation demands c_k and data sizes b_k; default b_k, c_k, and $T_d^{\max}$ from Table 1; random task dependency.

Comput. Cycles c_k Per Bit	Finish Time of Last Task (s) FT_K	Data Size b_k in KB	Finish Time FT_K of Last Task (s)
0	0	0	0
5	1.2155	50	1.6038
10	2.4201	100	3.248
15	3.6314	150	5.1438
20	4.8343	200	6.1856
25	6.0631	250	8.256
30	7.2743	300	9.9504
35	8.4917	350	11.5976
40	9.6840	400	12.9008

Figure 5a indicates that the ASO battery energy consumption is nearly constant with a slight increase. In the ASO case, the UE consumes battery energy for transmitting the data directly to the server. When the computations c_k per bit increase, the UE consumes slightly more energy for waiting for the server (however, the energy consumption for waiting is far lower compared to the transmission energy). For simple tasks (with small c_k), the task transmission to the MEC server (directly, ASO, or indirectly via helper relays, CMCO) consumes more UE and UH battery energy than executing the simple tasks locally (ALO) or on nearby helpers (CPCO). However, as tasks become more complicated (i.e., as c_k increases), the battery energy consumption for task execution on nearby helpers (CPCO) starts to exceed the battery energy consumption for transmitting the tasks via helpers (CMCO) to the MEC server.

We observe from Figure 5a that our proposed EETO method achieves the lowest average battery energy consumption compared to the other approaches. EETO trades optimally between the computation cooperation (CPCO) and the communication cooperation (CMCO) functionalities of the helper nodes, i.e., optimally trades off between all task execution options in Section 3.4, and thus achieves the minimal average battery energy consumption across the full range of task complexities, i.e., required computation cycles c_k per bit for task k. Importantly, in typically heterogeneous operating scenarios with a wide range of task complexities c_k (as further examined in Figure 8), our proposed EETO approach makes the optimal decision for each task k depending on the individual characteristics of each task k and thus can extract substantial energy consumption reductions compared to the CPCO or CMCO benchmarks, which could exploit only one type of cooperation.

Figure 5b shows that when the task data size b_k increases, more battery energy is required to offload the tasks to the helper or server, and more computation resources are required to complete the execution process. Therefore, the average battery energy consumption of all approaches increases as b_k increases. However, the EETO energy consumption is lower compared to the other methods for each data size b_k. The right part of Table 2 indicates that for increasing task data size b_k, ALO cannot satisfy the latency requirements, thus demonstrating again the need for an offloading method.

6.2.2. Impact of Transmission Power P_k^t and UE Speed

Figure 6a shows the average battery energy consumption as a function of the transmit power P_k^{tr}, indicting an overall growing trend of the average battery energy consumption with increasing P_k^{tr}. This is mainly because P_k^{tr} influences the transmission data rate logarithmically (see Equation (3)) and the battery energy consumption for task offloading linearly (see Equations (7), (10), and (13)). We observe from Figure 6a that for low $P_k^{tr} = 45$ mW, CMCO and EETO consume only about two-thirds of the battery energy of CPCO and less than half of ASO, mainly due to the energy-efficient relay task transmission to the MEC servers (however, a low P_k^{tr} reduces the offloading speed and makes it harder to meet the task deadlines). In contrast, for a high transmission power, EETO and CPCO achieve substantial battery energy savings compared to ASO and CMCO by avoiding the energy-expensive direct transmission to an MEC server and the relay transmission by a helper (whereby the CMCO transmits the task data twice, once from the UE to the UH and then from the UH to the MEC server).

Figure 6b shows the average battery energy consumption as a function of the UE speed, while the UH$_i$ are maintained at their default speeds which have an average of roughly 0.7 m/s in the Stanford data set [63]. We observe from Figure 6b an initially decreasing energy consumption as the UE speed increases to 1.5 m/s and then an increasing energy consumption as the UE speed increases above 1.5 m/s. This is mainly because a UE with a similar moving speed as the surrounding UH$_i$ has typically more UH$_i$ in its vicinity for a longer sojourn time. Importantly, Figure 6b indicates that EETO consistently reduces the battery energy consumption compared to CPCO and CMCO across the full range of considered UE speeds.

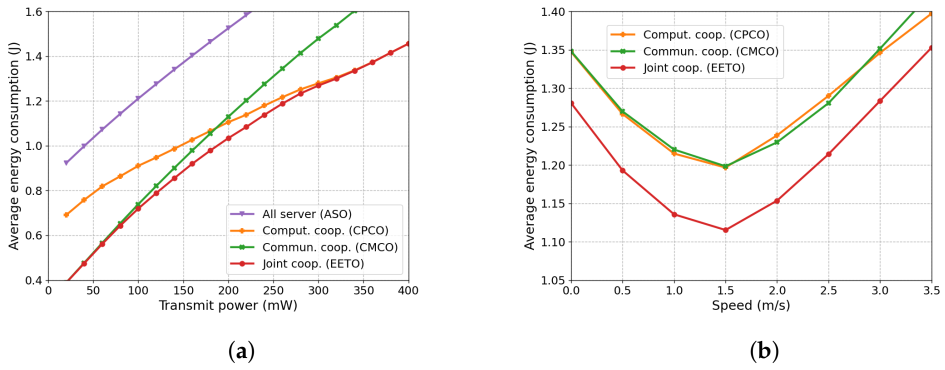


Figure 6. Average battery energy consumption $E_{\text{tot}}^{\text{exe}}$ vs. transmit power and UE speed; fixed parameters: b_k, c_k, $T_d^{\max}$, (and P_k^{tr} for plot (**b**)) from Table 1; random task dependency (UE speed 1 m/s for plot (**a**)). (**a**) Transmit power P_k^{tr}, (**b**) UE speed.

6.2.3. Impact of Task Dependency

Figure 7a,b show the finish time FT_K and average energy consumption $E_{\text{tot}}^{\text{exe}}$ of the five algorithms for various task dependency relationship graphs, namely sequential, random, and parallel. In the sequential dependency graph, each task k has one predecessor and to start the task execution, its predecessor task needs to be completed. However, in the random dependency graphs, each task k can have zero or multiple predecessor tasks, which requires totally different offloading decisions compared to the sequential dependency graph. Since we can execute multiple tasks at the same time with the random and parallel graphs, the execution time is less than for the sequential graph for all methods (for ALO, the sequential finish time is 21.2 s, i.e., outside the range plotted in Figure 7a). We can also see that EETO does not always have the shortest execution time since the aim is to minimize the energy while keeping the time within the deadline $T_d^{\max}$. However, the EETO execution time is very close to the shortest execution time.

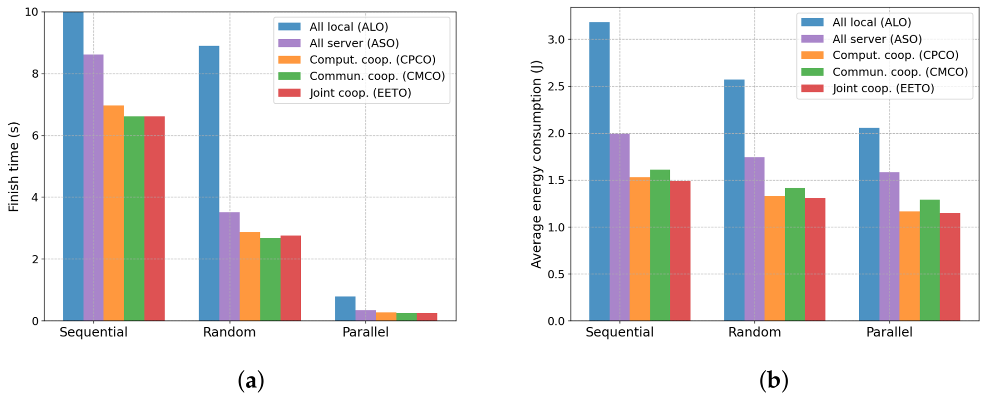


Figure 7. Impact of task dependency graph; fixed parameters: uniform random data size $b_k = 280$–320 KB, $c_k = 30$–50 cycles/bit (uniform random), task deadline $T_d^{\max} = 9$ s. (**a**) Finish time, (**b**) average battery energy consumption.

Focusing on the energy consumption, we observe from Figure 7b that the parallel task dependency leads to the lowest energy consumption. This reduction in energy consumption is mainly due to the reduced waiting time, i.e., the more independent the tasks are, the less energy is consumed for waiting for the completion of predecessor tasks. Overall, we observe from Figure 7b that among all methods and all task dependency cases, our proposed EETO achieves the lowest battery energy consumption.

6.2.4. Impact of Task Deadline $T_d^{\max}$

Figure 8 shows the average battery energy consumption as a function of the task completion deadline $T_d^{\max}$ for two different types of task heterogeneity. Figure 8a considers a wide range of computational complexities c_k of the individual tasks $k,\ k = 1, \ldots, K$, that are drawn uniformly randomly over the range [15, 55] cycles/bit with a prescribed latency requirement $T_d^{\max}$ for the entire set of K tasks. Figure 8b considers specified deadlines for tasks $k = 1$–7 of $FT_k = [0.2, 0.4, 0.4, 0.4, 0.6, 0.6, 0.6]$ s, while tasks $k = 8$–25 have the deadline $T_d^{\max}$ ranging from 2 to 4 s, for a narrower range of task complexities c_k.

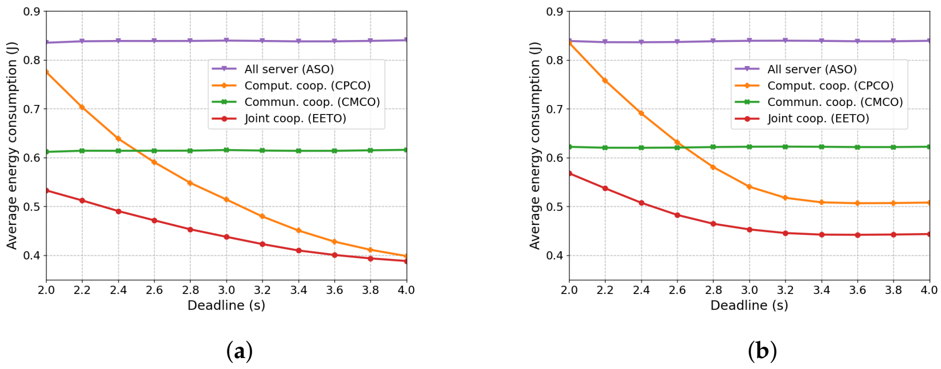


Figure 8. Average battery energy consumption vs. task execution deadline $T_d^{\max}$; fixed parameters: uniform random data size $b_k = 150$–180 KB, random task dependency. (**a**) Overall deadline $FT_K < T_d^{\max}$, $c_k = 15$–55 cycles/bit, (**b**) deadlines for ind. tasks FT_k, $c_k = 32$–38 cycles/bit.

ASO and CMCO execute all tasks on MEC servers within 2 s; thus, even if the deadline $T_d^{\max}$ increases, the energy consumption remains essentially constant. However, an increasing deadline $T_d^{\max}$ allows energy-efficient helpers to execute tasks, even if they have long processing times, thus reducing the CPCO and EETO energy consumption. By optimally trading off the task execution options, EETO achieves some moderate energy reductions for the heterogeneous task scenarios in Figure 8 compared to the minimum of the CMCO and CPCO energy consumptions. Importantly, EETO flexibly achieves the minimum energy consumption across the entire examined $T_d^{\max}$ range.

7. Conclusions

7.1. Summary of This Article

We developed and evaluated the novel Energy-Efficient Task Offloading (EETO) algorithm for dependent computation tasks in a cooperative edge computing setting with mobile end devices that contribute computation and communication relay support in a sliced edge network environment. EETO accommodates arbitrary task dependencies that are characterized by a general task dependency graph and employs a deep-learning-based trajectory prediction for the device sojourn times in the wireless transmission ranges. We have formulated the task offloading optimization problem as a quadratically constrained quadratic programming (QCQP) problem and developed a solution strategy that obtains the task offloading decisions through a semidefinite relaxation and stochastic mapping from the QCQP solution.

The simulation evaluations indicate the EETO consistently achieves low battery energy consumption across heterogeneous parameter settings and scenarios. In particular, the EETO substantially outperforms naive benchmarks that compute the tasks locally or offload all tasks to MEC servers. EETO also outperforms benchmarks with a limited set of offloading decision options; specifically, we considered benchmarks that allow helper nodes to only function as task processing (computations) nodes (CPCO) or to only function as task communication (relay) nodes for offloading to an MEC server (CMCO). We found that the CPCO and CMCO benchmarks attain the EETO performance in some scenarios, while in other scenarios, the EETO achieves significant performance gains.

7.2. Limitations and Future Research Directions

A limitation of the developed EETO algorithm is the computational effort for solving the QCQP, as noted in Section 5. One strategy for simplifying the QCQP could be to reduce the considered task offloading options for scenarios where a limited set of options achieve or nearly achieve the optimal EETO performance. For instance, a machine learning approach could learn the scenarios where a limited set of offloading options, such as the naive options or CPCO or CMCO, closely approach the EETO performance and then consider only the reduced set of offloading options in the solution of the optimization problem. Based on training data sets that can be created with the optimal EETO offloading decision making developed in this article, future research could investigate such simplified offloading strategies for scenarios that would need to be identified with pre-trained machine learning models.

Another strategy for simplifying the QCQP could be to "thin" out the set of potential helper nodes or MEC server nodes that are considered for task offloading, that is, the helper and MEC server nodes to be considered in the solution of the QCQP could be pre-selected according to criteria that indicate a high level of potential usefulness of a helper or MEC server. In scenarios with a high density of potential helper nodes in the vicinity of a UE that seeks to offload tasks, the helper nodes could be thinned out randomly. Alternatively, the pre-selection could be based on the distance from the UE to the helper and MEC server nodes or the wireless channel conditions. The pre-selection could be aided by machine learning strategies. For the development of such machine learning strategies, the formal optimization model and QCQP formulation and solution could be utilized to generate solution data sets for training. A related machine learning strategy could learn a direct mapping, i.e., a mapping from a given set of task properties as well as available communication and computing resources to a set of task offloading decisions. A model trained with solution data sets obtained from the QCQP solutions could potentially provide such a direct mapping with a simple neural network forward pass.

The present study has focused on developing a general mathematical model for the offloading of dependent computation tasks to mobile helper nodes, whereby the task properties as well as available communication and computation resources are characterized through parsimonious model parameters. A future research direction is to conduct end-to-end evaluations of the task offloading in testbeds with popular distributed computing applications, such as distributed data management and database operations [68,69], distributed Internet of Things data analytics [70,71], distributed video analytics [72–74], and general distributed artificial intelligence and data analytics [75,76]. Moreover, popular user-oriented applications, such as augmented and virtual reality [77–82] and online gaming [83], are important application domains to examine in the context of cooperative task offloading to sliced networked computing resources. Such testbed evaluations could examine the typical values of the model parameters, e.g., for the task complexity c_k, as well the impacts of real-world wireless bandwidth fluctuations and network congestion. Another important direction for future work is to investigate incentive mechanisms that promote fair cooperation between users.

Author Contributions: Conceptualization, M.M., G.P.K., X.G., M.R., and F.H.P.F.; methodology, M.M., S.S., G.P.K., X.G., M.R., and F.H.P.F.; software, M.M., S.S., and Y.H.; validation, M.M., S.S., V.L., and Y.H.; formal analysis, M.M., S.S., and M.R.; investigation, S.S. and Y.H.; resources, V.L. and F.H.P.F.; data curation, S.S., Y.H., and V.L.; writing—original draft preparation, M.M. and M.R.; writing—review and editing, M.M. and M.R.; visualization, V.L.; supervision, G.P.K., X.G., M.R., and F.H.P.F.; project administration, G.P.K., X.G., and F.H.P.F.; funding acquisition, G.P.K., X.G., and F.H.P.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the European Union H2020 research and innovation program under grant agreement H2020-MCSA-ITN-2016-SECRET 722424 [84] and in part by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy–EXC2050/1—Project ID 390696704—Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Fettweis, G.P. The tactile internet: Applications and challenges. *IEEE Veh. Technol. Mag.* **2014**, *9*, 64–70. [CrossRef]
2. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
3. Ateya, A.A.; Algarni, A.D.; Hamdi, M.; Koucheryavy, A.; Soliman, N.F. Enabling Heterogeneous IoT Networks over 5G Networks with Ultra-Dense Deployment—Using MEC/SDN. *Electronics* **2021**, *10*, 910. [CrossRef]
4. Zhang, Y.; Chen, G.; Du, H.; Yuan, X.; Kadoch, M.; Cheriet, M. Real-Time Remote Health Monitoring System Driven by 5G MEC-IoT. *Electronics* **2020**, *9*, 1753. [CrossRef]
5. Shariatmadari, H.; Ratasuk, R.; Iraji, S.; Laya, A.; Taleb, T.; Jäntti, R.; Ghosh, A. Machine-type communications: Current status and future perspectives toward 5G systems. *IEEE Commun. Mag.* **2015**, *53*, 10–17. [CrossRef]
6. Ahvar, E.; Ahvar, S.; Raza, S.M.; Manuel Sanchez Vilchez, J.; Lee, G.M. Next Generation of SDN in Cloud-Fog for 5G and Beyond-Enabled Applications: Opportunities and Challenges. *Network* **2021**, *1*, 28–49. [CrossRef]
7. Ruan, J.; Xie, D. Networked VR: State of the Art, Solutions, and Challenges. *Electronics* **2021**, *10*, 166. [CrossRef]
8. Kang, K.D.; Chen, L.; Yi, H.; Wang, B.; Sha, M. Real-Time Information Derivation from Big Sensor Data via Edge Computing. *Big Data Cogn. Comput.* **2017**, *1*, 5. [CrossRef]
9. Guan, S.; Boukerche, A. A Novel Mobility-aware Offloading Management Scheme in Sustainable Multi-access Edge Computing. *IEEE Trans. Sustain. Comput. Print* **2021**. [CrossRef]
10. Jin, Y.; Lee, H. On-Demand Computation Offloading Architecture in Fog Networks. *Electronics* **2019**, *8*, 1076. [CrossRef]
11. Lan, Y.; Wang, X.; Wang, C.; Wang, D.; Li, Q. Collaborative Computation Offloading and Resource Allocation in Cache-Aided Hierarchical Edge-Cloud Systems. *Electronics* **2019**, *8*, 1430. [CrossRef]
12. Ouyang, T.; Zhou, Z.; Chen, X. Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2333–2345. [CrossRef]
13. Zhu, T.; Shi, T.; Li, J.; Cai, Z.; Zhou, X. Task scheduling in deadline-aware mobile edge computing systems. *IEEE Internet Things J.* **2018**, *6*, 4854–4866. [CrossRef]
14. Hu, Y.C.; Patel, M.; Sabella, D.; Sprecher, N.; Young, V. Mobile edge computing—A key technology towards 5G. *ETSI White Pap.* **2015**, *11*, 1–16.
15. Oliveira, A., Jr.; Cardoso, K.; Sousa, F.; Moreira, W. A Lightweight Slice-Based Quality of Service Manager for IoT. *IoT* **2020**, *1*, 49–75. [CrossRef]
16. Nadeem, L.; Azam, M.A.; Amin, Y.; Al-Ghamdi, M.A.; Chai, K.K.; Khan, M.F.N.; Khan, M.A. Integration of D2D, Network Slicing, and MEC in 5G Cellular Networks: Survey and Challenges. *IEEE Access* **2021**, *9*, 37590–37612. [CrossRef]
17. Bellavista, P.; Chessa, S.; Foschini, L.; Gioia, L.; Girolami, M. Human-enabled edge computing: Exploiting the crowd as a dynamic extension of mobile edge computing. *IEEE Commun. Mag.* **2018**, *56*, 145–155. [CrossRef]
18. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H. Device-enhanced MEC: Multi-access edge computing (MEC) aided by end device computation and caching: A survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
19. Ali, Z.; Khaf, S.; Abbas, Z.H.; Abbas, G.; Muhammad, F.; Kim, S. A Deep Learning Approach for Mobility-Aware and Energy-Efficient Resource Allocation in MEC. *IEEE Access* **2020**, *8*, 179530–179546. [CrossRef]
20. Chen, S.; Zheng, Y.; Lu, W.; Varadarajan, V.; Wang, K. Energy-Optimal Dynamic Computation Offloading for Industrial IoT in Fog Computing. *IEEE Trans. Green Commun. Netw.* **2020**, *4*, 566–576. [CrossRef]
21. Khan, P.W.; Abbas, K.; Shaiba, H.; Muthanna, A.; Abuarqoub, A.; Khayyat, M. Energy Efficient Computation Offloading Mechanism in Multi-Server Mobile Edge Computing—An Integer Linear Optimization Approach. *Electronics* **2020**, *9*, 1010. [CrossRef]
22. Tang, L.; Hu, H. Computation Offloading and Resource Allocation for the Internet of Things in Energy-Constrained MEC-Enabled HetNets. *IEEE Access* **2020**, *8*, 47509–47521. [CrossRef]
23. Zhang, T.; Chen, W. Computation Offloading in Heterogeneous Mobile Edge Computing with Energy Harvesting. *IEEE Trans. Green Commun. Netw.* **2021**, *5*, 552–565. [CrossRef]
24. Mehrabi, M.; Shen, S.; Latzko, V.; Wang, Y.; Fitzek, F. Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks. In Proceedings of the GLOBECOM 2020–2020 IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; pp. 1–6.
25. Anajemba, J.H.; Yue, T.; Iwendi, C.; Alenezi, M.; Mittal, M. Optimal Cooperative Offloading Scheme for Energy Efficient Multi-Access Edge Computation. *IEEE Access* **2020**, *8*, 53931–53941. [CrossRef]
26. Jia, Q.; Xie, R.; Tang, Q.; Li, X.; Huang, T.; Liu, J.; Liu, Y. Energy-efficient computation offloading in 5G cellular networks with edge computing and D2D communications. *IET Commun.* **2019**, *13*, 1122–1130. [CrossRef]
27. Ouyang, W.; Chen, Z.; Wu, J.; Yu, G.; Zhang, H. Dynamic Task Migration Combining Energy Efficiency and Load Balancing Optimization in Three-Tier UAV-Enabled Mobile Edge Computing System. *Electronics* **2021**, *10*, 190. [CrossRef]
28. Qiao, G.; Leng, S.; Zhang, Y. Online learning and optimization for computation offloading in D2D edge computing and networks. *Mob. Netw. Appl.* **2021**, in print. [CrossRef]

29. Cui, K.; Lin, B.; Sun, W.; Sun, W. Learning-Based Task Offloading for Marine Fog-Cloud Computing Networks of USV Cluster. *Electronics* **2019**, *8*, 1287. [CrossRef]
30. Xing, H.; Liu, L.; Xu, J.; Nallanathan, A. Joint task assignment and resource allocation for D2D-enabled mobile-edge computing. *IEEE Trans. Commun.* **2019**, *67*, 4193–4207. [CrossRef]
31. Ranji, R.; Mansoor, A.M.; Sani, A.A. EEDOS: An energy-efficient and delay-aware offloading scheme based on device to device collaboration in mobile edge computing. *Telecommun. Syst.* **2020**, *73*, 171–182. [CrossRef]
32. Saleem, U.; Liu, Y.; Jangsher, S.; Tao, X.; Li, Y. Latency minimization for D2D-enabled partial computation offloading in mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 4472–4486. [CrossRef]
33. Sun, M.; Xu, X.; Tao, X.; Zhang, P. Large-scale user-assisted multi-task online offloading for latency reduction in D2D-enabled heterogeneous networks. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 2456–2467. [CrossRef]
34. Yang, Y.; Long, C.; Wu, J.; Peng, S.; Li, B. D2D-Enabled Mobile-Edge Computation Offloading for Multi-user IoT Network. *IEEE Internet Things J.* **2021**, in print.
35. Fan, N.; Wang, X.; Wang, D.; Lan, Y.; Hou, J. A Collaborative Task Offloading Scheme in D2D-Assisted Fog Computing Networks. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference (WCNC), Seoul, Korea, 25–28 May 2020.
36. Ko, H.; Pack, S. Distributed Device-to-Device Offloading System: Design and Performance Optimization. *IEEE Trans. Mob. Comput.* **2021**, in print. [CrossRef]
37. Shi, T.; Cai, Z.; Li, J.; Gao, H. CROSS: A crowdsourcing based sub-servers selection framework in D2D enhanced MEC architecture. In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 1134–1144.
38. Sun, M.; Xu, X.; Huang, Y.; Wu, Q.; Tao, X.; Zhang, P. Resource Management for Computation Offloading in D2D-Aided Wireless Powered Mobile-Edge Computing Networks. *IEEE Internet Things J.* **2021**, *8*, 8005–8020. [CrossRef]
39. Tong, M.; Wang, X.; Wang, Y.; Lan, Y. Computation Offloading Scheme with D2D for MEC-enabled Cellular Networks. In Proceedings of the 2020 IEEE/CIC International Conference on Communications in China (ICCC Workshops), Xiamen, China, 28–30 July 2020; pp. 111–116.
40. Wang, J.; Wu, W.; Liao, Z.; Jung, Y.W.; Kim, J.U. An Enhanced PROMOT Algorithm with D2D and Robust for Mobile Edge Computing. *J. Internet Technol.* **2020**, *21*, 1437–1445.
41. Wan, L. Computation Offloading Strategy Based on Improved Auction Model in Mobile Edge Computing Network. *J. Phys. Conf. Ser.* **2021**, *1852*, 042047. [CrossRef]
42. Zhang, X. Enhancing mobile cloud with social-aware device-to-device offloading. *Comput. Commun.* **2021**, *168*, 1–11. [CrossRef]
43. Chen, X.; Zhou, Z.; Wu, W.; Wu, D.; Zhang, J. Socially-motivated cooperative mobile edge computing. *IEEE Netw.* **2018**, *32*, 177–183. [CrossRef]
44. Li, Z.; Hu, H.; Hu, H.; Huang, B.; Ge, J.; Chang, V. Security and Energy-aware Collaborative Task Offloading in D2D communication. *Future Gener. Comput. Syst.* **2021**, *118*, 358–373. [CrossRef]
45. Baek, H.; Ko, H.; Pack, S. Privacy-Preserving and Trustworthy Device-to-Device (D2D) Offloading Scheme. *IEEE Access* **2020**, *8*, 191551–191560. [CrossRef]
46. Zhang, Y.; Qin, X.; Song, X. Mobility-Aware Cooperative Task Offloading and Resource Allocation in Vehicular Edge Computing. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Seoul, Korea, 25–28 May 2020.
47. Zhu, C.; Tao, J.; Pastor, G.; Xiao, Y.; Ji, Y.; Zhou, Q.; Li, Y.; Ylä-Jääski, A. Folo: Latency and Quality Optimized Task Allocation in Vehicular Fog Computing. *IEEE Internet Things J.* **2019**, *6*, 4150–4161. [CrossRef]
48. Misra, S.; Bera, S. Soft-VAN: Mobility-Aware Task Offloading in Software-Defined Vehicular Network. *IEEE Trans. Veh. Technol.* **2020**, *69*, 2071–2078. [CrossRef]
49. Selim, M.M.; Rihan, M.; Yang, Y.; Ma, J. Optimal task partitioning, Bit allocation and trajectory for D2D-assisted UAV-MEC systems. *Peer-Netw. Appl.* **2021**, *14*, 215–224. [CrossRef]
50. Zhang, L.; Zhang, Z.Y.; Min, L.; Tang, C.; Zhang, H.Y.; Wang, Y.H.; Cai, P. Task Offloading and Trajectory Control for UAV-Assisted Mobile Edge Computing Using Deep Reinforcement Learning. *IEEE Access* **2021**, *9*, 53708–53719. [CrossRef]
51. Wu, C.; Peng, Q.; Xia, Y.; Lee, J. Mobility-Aware Tasks Offloading in Mobile Edge Computing Environment. In Proceedings of the 2019 Seventh International Symposium on Computing and Networking (CANDAR), Nagasaki, Japan, 26–29 November 2019; pp. 204–210.
52. Jeon, Y.; Baek, H.; Pack, S. Mobility-Aware Optimal Task Offloading in Distributed Edge Computing. In Proceedings of the 2021 International Conference on Information Networking (ICOIN), Bangkok, Thailand, 12–15 January 2021; pp. 65–68.
53. Wang, D.; Liu, Z.; Wang, X.; Lan, Y. Mobility-Aware Task Offloading and Migration Schemes in Fog Computing Networks. *IEEE Access* **2019**, *7*, 43356–43368. [CrossRef]
54. Saleem, U.; Liu, Y.; Jangsher, S.; Li, Y.; Jiang, T. Mobility-Aware Joint Task Scheduling and Resource Allocation for Cooperative Mobile Edge Computing. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 360–374. [CrossRef]
55. Feng, Z.; Zhu, Y. A survey on trajectory data mining: Techniques and applications. *IEEE Access* **2016**, *4*, 2056–2067. [CrossRef]
56. Wang, Z.; Zhao, Z.; Min, G.; Huang, X.; Ni, Q.; Wang, R. User mobility aware task assignment for mobile edge computing. *Future Gener. Comput. Syst.* **2018**, *85*, 1–8. [CrossRef]
57. Liu, F.; Huang, Z.; Wang, L. Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for IoT sensors. *Sensors* **2019**, *19*, 1105. [CrossRef]

58. Liu, H.; Eldarrat, F.; Alqahtani, H.; Reznik, A.; De Foy, X.; Zhang, Y. Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Syst. J.* **2017**, *12*, 2495–2508. [CrossRef]
59. Sarmiento, D.E.; Lebre, A.; Nussbaum, L.; Chari, A. Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 256–281. [CrossRef]
60. Mahmoodi, S.E.; Uma, R.; Subbalakshmi, K. Optimal joint scheduling and cloud offloading for mobile applications. *IEEE Trans. Cloud Comput.* **2019**, *7*, 301–313. [CrossRef]
61. Hao, W.; Yang, S. Small cell cluster-based resource allocation for wireless backhaul in two-tier heterogeneous networks with massive MIMO. *IEEE Trans. Veh. Technol.* **2017**, *67*, 509–523. [CrossRef]
62. Song, C.; Qu, Z.; Blumm, N.; Barabási, A.L. Limits of predictability in human mobility. *Science* **2010**, *327*, 1018–1021. [CrossRef]
63. Mangalam, K.; Girase, H.; Agarwal, S.; Lee, K.H.; Adeli, E.; Malik, J.; Gaidon, A. It is not the journey but the destination: Endpoint conditioned trajectory prediction. In *European Conference on Computer Vision*; Lecture Notes in Computer Science; Vedaldi, A., Bischof, H., Brox, T., Frahm, J., Eds.; Springer: Cham, Switzerland, 2020; Volume 12347, pp. 759–776.
64. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*; Springer: Berlin/Heidelberg, Germany, 1972; pp. 85–103.
65. O'Donoghue, B.; Chu, E.; Parikh, N.; Boyd, S. Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optim. Theory Appl.* **2016**, *169*, 1042–1068. [CrossRef]
66. Deng, L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* **2020**, *108*, 485–532. [CrossRef]
67. Shantharama, P.; Thyagaturu, A.S.; Reisslein, M. Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies. *IEEE Access* **2020**, *8*, 132021–132085. [CrossRef]
68. Liu, C.; Jiang, H.; Paparrizos, J.; Elmore, A.J. Decomposed bounded floats for fast compression and queries. *Proc. VLDB Endow.* **2021**, *14*, 2586–2598.
69. Paparrizos, J.; Liu, C.; Barbarioli, B.; Hwang, J.; Edian, I.; Elmore, A.J.; Franklin, M.J.; Krishnan, S. VergeDB: A database for IoT analytics on edge devices. In Proceedings of the 11th Annual Conference on Innovative Data Systems Research (CIDR '21), Chaminade, CA, USA, 10–13 January 2021; pp. 1–8.
70. Happ, D.; Bayhan, S.; Handziski, V. JOI: Joint placement of IoT analytics operators and pub/sub message brokers in fog-centric IoT platforms. *Future Gener. Comput. Syst.* **2021**, *119*, 7–19. [CrossRef]
71. Marah, B.D.; Jing, Z.; Ma, T.; Alsabri, R.; Anaadumba, R.; Al-Dhelaan, A.; Al-Dhelaan, M. Smartphone architecture for edge-centric IoT analytics. *Sensors* **2020**, *20*, 892. [CrossRef]
72. Deng, Y.; Han, T.; Ansari, N. FedVision: Federated video analytics with edge computing. *IEEE Open J. Comput. Soc.* **2020**, *1*, 62–72. [CrossRef]
73. Ali, M.; Anjum, A.; Rana, O.; Zamani, A.R.; Balouek-Thomert, D.; Parashar, M. RES: Real-time video stream analytics using edge enhanced clouds. *IEEE Trans. Cloud Comput.* **2021**, in print. [CrossRef]
74. Rocha Neto, A.; Silva, T.P.; Batista, T.; Delicato, F.C.; Pires, P.F.; Lopes, F. Leveraging edge intelligence for video analytics in smart city applications. *Information* **2021**, *12*, 14. [CrossRef]
75. Krishnan, S.; Elmore, A.J.; Franklin, M.; Paparrizos, J.; Shang, Z.; Dziedzic, A.; Liu, R. Artificial intelligence in resource-constrained and shared environments. *ACM SIGOPS Oper. Syst. Rev.* **2019**, *53*, 1–6. [CrossRef]
76. Paparrizos, J.; Franklin, M.J. GRAIL: Efficient time-series representation learning. *Proc. VLDB Endow.* **2019**, *12*, 1762–1777. [CrossRef]
77. Alshahrani, A.; Elgendy, I.A.; Muthanna, A.; Alghamdi, A.M.; Alshamrani, A. Efficient multi-player computation offloading for VR edge-cloud computing systems. *Appl. Sci.* **2020**, *10*, 5515. [CrossRef]
78. Ameur, A.B.; Araldo, A.; Bronzino, F. On the deployability of augmented reality using embedded edge devices. In Proceedings of the 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2021; pp. 1–6.
79. Braud, T.; Zhou, P.; Kangasharju, J.; Hui, P. Multipath computation offloading for mobile augmented reality. In Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications (PerCom), Austin, TX, USA, 23–27 March 2020; pp. 1–10.
80. Chen, M.; Liu, W.; Wang, T.; Liu, A.; Zeng, Z. Edge intelligence computing for mobile augmented reality with deep reinforcement learning approach. *Comput. Netw.* **2021**, *195*, 108186. [CrossRef]
81. Ren, P.; Qiao, X.; Huang, Y.; Liu, L.; Dustdar, S.; Chen, J. Edge-assisted distributed DNN collaborative computing approach for mobile web augmented reality in 5G networks. *IEEE Netw.* **2020**, *34*, 254–261. [CrossRef]
82. Vidal-Balea, A.; Blanco-Novoa, O.; Picallo-Guembe, I.; Celaya-Echarri, M.; Fraga-Lamas, P.; Lopez-Iturri, P.; Azpilicueta, L.; Falcone, F.; Fernández-Caramés, T.M. Analysis, design and practical validation of an augmented reality teaching system based on microsoft HoloLens 2 and edge computing. *Eng. Proc.* **2020**, *2*, 52.
83. Yang, X.; Luo, H.; Sun, Y.; Obaidat, M.S. Energy-efficient collaborative offloading for multiplayer games with cache-aided MEC. In Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7.
84. Rodriguez, J.; Radwan, A.; Barbosa, C.; Fitzek, F.H.P.; Abd-Alhameed, R.A.; Noras, J.M.; Jones, S.M.R.; Politis, I.; Galiotos, P.; Schulte, G.; et al. SECRET–Secure network coding for reduced energy next generation mobile small cells: A European Training Network in wireless communications and networking for 5G. In Proceedings of the 2017 Internet Technologies and Applications (ITA), Wrexham, UK, 12–15 September 2017; pp. 329–333.