Adaptive reverse task offloading in edge computing for AI processes

Amanatidis, Petros; Karampatzakis, Dimitris; Michailidis, Georgios; Lagkas, Thomas; Iosifidis, George

DOI
10.1016/j.comnet.2024.110844

Publication date
2024

Document Version
Final published version

Published in
Computer Networks

Citation (APA)
Amanatidis, P., Karampatzakis, D., Michailidis, G., Lagkas, T., & Iosifidis, G. (2024). Adaptive reverse task offloading in edge computing for AI processes. *Computer Networks*, *255*, Article 110844. https://doi.org/10.1016/j.comnet.2024.110844

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Adaptive reverse task offloading in edge computing for AI processes

Petros Amanatidis [a], Dimitris Karampatzakis [a], Georgios Michailidis [a], Thomas Lagkas [a,*], George Iosifidis [b]

[a] Department of Informatics, Democritus University of Thrace, Kavala 65404, Greece
[b] Delft University of Technology, Delft 2628 XE, Netherlands

A R T I C L E I N F O

Keywords:
Task offloading
Optimization
Edge computing
Resource allocation
AI processes

A B S T R A C T

Nowadays, we witness the proliferation of edge IoT devices, ranging from smart cameras to autonomous vehicles, with increasing computing capabilities, used to implement AI-based services in users' proximity, right at the edge. As these services are often computationally demanding, the popular paradigm of offloading their tasks to nearby cloud servers has gained much traction and been studied extensively. In this work, we propose a new paradigm that departs from the above typical edge computing offloading idea. Namely, we argue that it is possible to leverage these end nodes to assist larger nodes (e.g., cloudlets) in executing AI tasks. Indeed, as more and more end nodes are deployed, they create an abundance of idle computing capacity, which, when aggregated and exploited in a systematic fashion, can be proved beneficial. We introduce the idea of reverse offloading and study a scenario where a powerful node splits an AI task into a group of subtasks and assigns them to a set of nearby edge IoT nodes. The goal of each node is to minimize the overall execution time, which is constrained by the slowest subtask, while adhering to predetermined energy consumption and AI performance constraints. This is a challenging MINLP (Mixed Integer Non-Linear Problem) optimization problem that we tackle with a novel approach through our newly introduced EAI-ARO (Edge AI-Adaptive Reverse Offloading) algorithm. Furthermore, a demonstration of the efficacy of our reverse offloading proposal using an edge computing testbed and a representative AI service is performed. The findings suggest that our method optimizes the system's performance significantly when compared with a greedy and a baseline task offloading algorithm.

1. Introduction

The rapid development of computing resources and the increasing storage capacities of IoT devices has helped to develop as well as improve many applications. This has the effect of moving data processing from centralized cloud environments to decentralized edge computing [1,2]. Today there is an increasing number of services that require the collection and processing of data so as to extract valuable information. Indeed, the term edge intelligence has been recently coined to describe a gamut of applications that follow this paradigm [3–6].

Given the computation load and the need for specialized ML libraries, a prominent paradigm for the implementation of these services is edge computing, where the end devices (far edge) offload the most demanding of these tasks to nearby servers, e.g., cloudlets. In the last several years a remarkable number of studies explored the potential, and the different facets, of such computation offloading architectures [7–10]. One aspect that has started to gain a lot of traction is that these far-edge devices when in large numbers, can also serve as computation offloading terminals [11]. In fact, as more and more of these devices are deployed (billions of IoT nodes are expected) it is only reasonable to consider scenarios where larger nodes (at the edge or farther) send their tasks (perhaps after splitting them) to the far-edge for execution.

Several researchers have worked on task offloading [12–14], most of them offloading tasks from end devices to servers for processing. Moreover, contributions in the development of different optimization methods to minimize power consumption and latency while maximizing the performance of edge computing applications have been presented [15–17]. On the contrary, "reverse" offloading, i.e., offloading tasks from the cloudlets to the end devices (see Fig. 1) has not been thoroughly studied.

In this work, this reverse offloading idea, namely offloading from the server to the end devices, is proposed based on the formulation of an optimization problem for the overall performance. Our goal is to minimize the task completion time, which is defined as the longest subtask time, while considering accuracy and energy constraints. Considering that the end devices have heterogeneous computing times

* Corresponding author.
E-mail address: tlagkas@cs.duth.gr (T. Lagkas).

https://doi.org/10.1016/j.comnet.2024.110844
Received 18 April 2024; Received in revised form 1 September 2024; Accepted 3 October 2024
Available online 9 October 2024

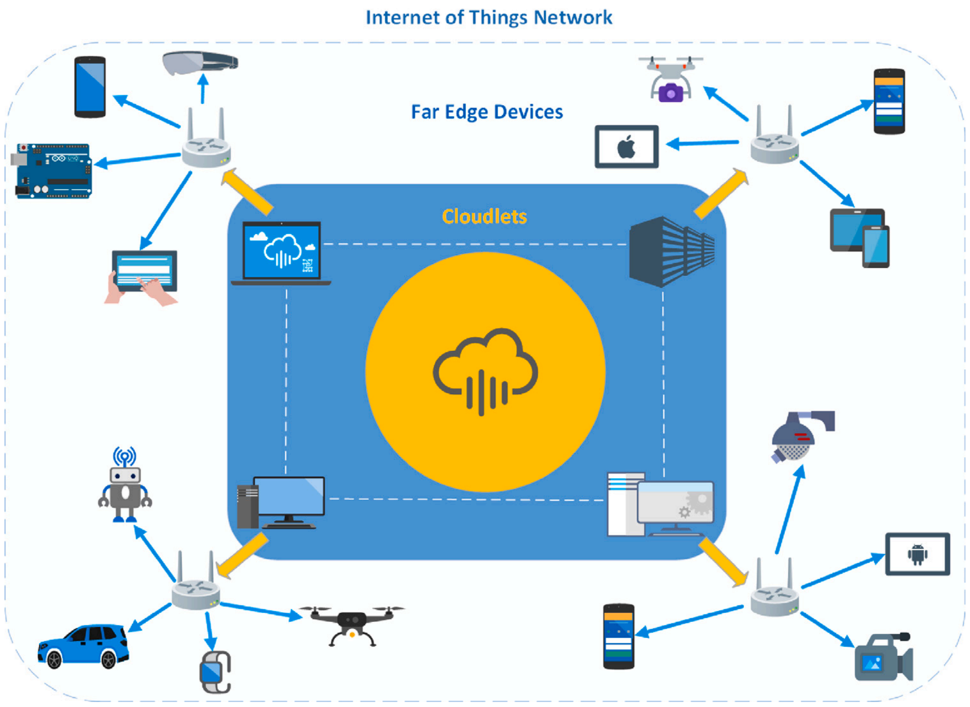


Fig. 1. Reverse task offloading proposal.

because of the dynamic network parameters and different deep learning models implemented, our optimization problem becomes challenging.

To provide a more specific definition of our optimization problem, we first optimize the task distribution percentage to minimize the latency under energy consumption constraints. Then, we enrich the optimization problem providing the flexibility to end-devices to choose from a set of available deep learning models that are used for object detection services on images. This choice affects the latency, the consumed energy, as well as the mean accuracy of the system which is added as a constraint in the optimization problem. Numerically, this extra degree of freedom inserts a discrete variable in the optimization problem, thus making continuous optimization algorithms unsuitable. We solve this optimization problem using a novel adaptive offloading approach that combines stochastic algorithms, namely Genetic Algorithms (GA), with Linear Programming (LP). Finally, we also examine the behavior of the algorithms in large-scale scenarios and propose dedicated strategies. The outcomes demonstrate significant cost savings and improvements in performance.

1.1. Methodology and contributions

We examine a wireless edge computing network model featuring nodes with diverse object detector models and a single access point. Within this network, each end device generates computer vision tasks, capable of local execution with specific accuracy and delay requirements. The end devices are interconnected via links with different capacities, consuming energy for the computation tasks.

To be more precise, we investigate a method where an edge server distributes the workload among several end devices to optimize the processing speed of AI workloads. To illustrate this service, we focus on processing images (e.g., videos with a specific duration) that are sent from edge servers to the end devices, on which an object detector is applied to locate the objects of interest. Each end device possesses an object detector with a different characteristic, namely the neural network size, which practically means that every device disposes different performance and efficiency. The edge server algorithm decides the distribution or splitting of the images to the end devices. Making this decision involves evaluating the potential performance gains, which are measured in terms of latency. For this purpose, we suggest employing different optimization algorithms implemented in the edge server, to improve the performance of such IoT networks.

Taking the conducted evaluation a step further, we develop comprehensive solutions implemented as “reverse” offloading algorithms. These solutions are evaluated through a series of experiments conducted on a wireless testbed consisting of 4 Raspberry Pis (RPis), along with the use of state-of-the-art object detector applications. Additionally, we leverage these experimental measurements to simulate larger networks, or more specifically we perform a numerical validation in a large-scaled testbed to illustrate the efficiency of the proposed methods. The results prove that the latency has been significantly reduced by our implemented strategy when compared to a greedy algorithm as used in [18–20] and a fixed task offloading baseline algorithm [21]. Moreover, our offloading method, which we call EAI-ARO, tailors its operation by taking resource availability into account. Therefore, this work adds the following contributions:

- **Performance Optimization**. We propose the idea of reverse offloading where powerful edge nodes split their tasks and assign the subtasks to far-edge nodes. This is of great importance when it comes to edge computing services and IoT networks. The contribution of this paper consists in the formulation of optimization problems for finding the minimum-time assignment strategy while respecting accuracy and energy constraints. We design an adaptive reverse offloading algorithm (EAI-ARO) for the above problem which is adaptable to various AI tasks, diverse edge devices with varying hardware and processing capabilities, as well as resource limitations.
- **Distribution Strategies**. We propose strategies for the distribution of the total task load based on the dynamic network conditions and the number of end devices. More specifically, when the number of end-devices turns the solution of the optimization problem to become too costly, we split the end-devices into clusters or more specifically into groups of devices and we optimize the distribution in each of these clusters separately. Moreover, we examine how the network conditions impact the frequency of optimization runs, i.e., the batch-splitting step.

The proposed architecture and decision framework is evaluated in our wireless testbed with a common deep learning (object detection) task and a benchmark COCO dataset [22]. The experiments show that our proposed methods achieve considerable cost savings and performance gains. Furthermore, the results also illustrate that the EAI-ARO algorithm can be implemented in various edge computing topologies with many end devices, different processing capabilities and different kinds of AI tasks.

The rest of the paper is structured as follows: In Section 2, we focus on exploring the literature. An introduction to the formulation of the optimization problems along with the proposed algorithms is provided in Section 3. Then, we present the results of our distribution mechanisms in Section 4. Last, in Section 5 we conclude our work.

2. Related work

This section presents different task offloading optimization approaches in edge computing networks used in recent literature.

Computation Offloading. Task offloading involves identifying specific tasks or workloads that can be transferred from one device to another, "traditionally" from far edge to edge nodes. For instance the study [23] explored the domain of Multi-Access Edge Computing (MEC) enhanced by 5G technology. One effective method according to the authors in reducing latency was to offload computing tasks from end devices to edge servers within the edge network. This work introduced a Particle Swarm Optimization (PSO) based task offloading technique, which provided low latency and energy efficiency by optimizing the offloading process. In [24], the problem of optimizing computation offloading and resource allocation in a dynamic multi-user (MEC) system was tackled. The authors want to minimize the total energy consumption while taking into account the uncertain resource demands of different tasks and delay constraints. They formulate the problem as a MINLP problem and propose a Q-learning-based reinforcement learning method to solve it. To address the performance limitations of mobile devices (MDs) with limited battery and computing resources, the authors in [25] addressed the problem of optimizing computational offloading in MEC networks. By offloading tasks to neighboring mobile edge servers (MES), the authors' fast and energy-efficient deep learning-based offloading technique (EFDOT) seeks to minimize overall costs, including service delay e and energy consumption. Real-time mobile application processing by investigating a MEC network opened new possibilities by Unmanned Aerial Vehicles (UAVs) [26]. The authors emphasize on minimizing system-wide computation costs in a dynamic environment where users generate tasks with time-varying probabilities by optimizing both edge server deployment and multi-user computation offloading. All these studies show the gains of the "traditional" offloading approaches attempted to optimize some of the objectives our work tries to optimize like minimizing the latency of the task execution, all with different algorithms like the PSO or with deep learning approaches. An approach that has been overlooked is the "reverse" offloading namely offloading from cloudlets to far edge devices. Furthermore, our work introduced a novel optimization method designed to enhance the efficiency of this reverse offloading approach.

Cooperative offloading. This approach emphasizes collaboration between the far edge and edge nodes. Collaborative caching in an EC context has been the subject of several researchers [27], with the goal of reducing the overall system cost, which includes costs for quality of service (QoS) and data migration overhead. To address this complex problem, an online solution based on Lyapunov optimality conditions was devised, called the collaborative edge data caching problem (CEDC). In [28] the issue of computational complexity and resource constraints associated with artificial intelligence (AI) algorithms, which present many difficulties in augmented reality systems, was studied. However, the rise of MEC offers a potentially effective way to tackle these problems. In order to improve recognition accuracy, minimize inference delays, and consume less energy, this paper introduced a framework for video-based AI inference processes in MEC systems. To this end, a MINLP problem was formulated. The inference complexity model and the accuracy model were derived and improved through experimentation, leading to an iterative solution approach using alternating optimization. In [29] an attempt to address privacy concerns in MEC systems by increasingly adopting federated learning (FL) is presented. The challenge lies in finding a balance between the learning accuracy of FL and energy consumption usage. To address this trade-off, this paper presented the FL-TD3 framework, specifically designed for large spaces of states and energies in a continuous domain. A framework [15] was introduced to enable collaborative task execution in Internet of Things (IoT) networks with limited resources, specifically addressing challenges in edge data analytics. The authors proposed an auction-based algorithm to optimize task-node assignments, ensuring optimal execution accuracy and minimal delays. Another, cooperative task execution approach in [30] ensures the effective transfer of edge tasks to nearby IoT devices and the efficient execution of local IoT tasks. The authors highlighted the significance of two conditions for IoT-assisted edge computing to succeed: maintaining the integrity of local IoT tasks and maximizing the use of computing resources to increase edge service throughput. Finally, all the above works execute different tasks like AI tasks or augmented reality tasks in a collaborative fashion while offloading from the far edge to the edge nodes (eg., edge servers). Our work has a different approach while having a non-cooperative fashion by executing the AI tasks sent from the cloudlet locally on the far edge device. The key component of this approach lies in finding the optimal task splitting decision according to the current dynamic systems parameters.

Reverse offloading. Offloading from cloudlets to far-edge devices has not been thoroughly studied, although some attempts to implement reverse offloading have been conducted especially in VEC (Vehicular Edge Computing). In [11] a reverse offloading framework was presented to reduce the amount of computational workloads on VEC servers by offloading the vehicular computations to Cooperative Vehicle-Infrastructure Systems (CVIS). This framework attempted to minimize the latency of processing the workloads by optimizing the reverse offloading and resource allocation. To address the challenge of limited computational resources of VEC servers, because of the growing amount of sensor data generated, another reverse computing offloading framework making use of the computer power of vehicles was introduced [31]. In this method offloading decision-making and resource allocation have been studied, including Q-Learning-based algorithms and constrained Markov decision processes. It is clear that not many studies have been conducted which highlight the reverse offloading approach, especially when handling AI workloads. In this work, we try to enhance this approach for AI applications, such as Deep Learning, which have not been examined thoroughly in the last years.

In conclusion, while recent literature provides valuable information on various aspects of edge computing, task offloading, and optimization methods, a particular need that our work studies derives from the gaps in the literature on reverse offloading. Our work contributes significantly to filling these gaps by presenting new reverse task offloading algorithms that specifically address the challenges of latency minimization while taking into account accuracy and energy consumption constraints in the context of artificial intelligence tasks, showing their effectiveness in practical scenarios through experimentation in a wireless testbed.

3. Model and problem formulation

3.1. Optimal task distribution

We introduce our system model and the mathematical formulation of the respective problem. In Table 1, we show all the key parameters and decision variables we use. The system consists of a set $\mathcal{N}$ of N devices for which a batch of images B or else the Batch step is sent for processing from the edge server. The task to be executed by the devices is an AI task or more precisely an object detection task. Every device i in our system implements a deep learning model (neural network) for object detection. The models may be of different size, thus differ in their computational performance, average accuracy and energy consumption (see Table 3). For every device $i \in \mathcal{N}$, the neural network used is described via its index $y_i \in \mathcal{M}$, where $\mathcal{M} = \{1, 2, \dots, M\}$ is a set of M neural networks.

Our purpose is to obtain a system with different processing capabilities to reveal sharp trade-offs. In more detail, the devices of our system implement the Yolov8 [32] object detector. The devices with different characteristics and configurations provide in real-time the confidence (accuracy) of the detection executed on the transmitted images by the edge server.

For each end device i, we introduce an optimization variable $x_i \in [0, 1]$, which represents the portion of B sent to the end device $i \in [1, N]$. We gather all these variables in a vector $\mathbf{x} = [x_1, \dots, x_N] \in [0, 1]^N$. Our goal is to minimize the maximum latency, i.e., the time to process all the images sent from the server to the end devices. For each device the latency is defined here as the total time needed for the communication with the server (T^i_{tx}) and the execution of the object detection process (T^i_{dl}), i.e.,:

$$L_i(x) = x_i B T^i_{tx} + x_i B T^i_{dl}(y_i) = C_i x_i,$$

where:

$$C_i = B(T^i_{tx} + T^i_{dl}(y_i)).$$

Moreover, in real-life applications, one further needs to take into account the available energy of each device to ensure that the device has actually the capacity to perform the attributed tasks. This adds an extra constraint for each device, s.t.:

$$E_i(x_i, y_i) \le E^i_{available},$$

where:

$$E_i(x_i, y_i) = x_i B\, E(y_i)$$

and $E(y_i)$ represents the energy consumed for the object detection process of one image by the neural network y_i.

Then, the optimization problem can be expressed as follows:

$$\begin{aligned} & \min_x \max_i C_i x_i \\ \text{s.t.} \quad & E_i(x_i, y_i) \le E^i_{available}, \quad i = 1, \dots, N \\ & \textstyle\sum_{i=1}^N x_i = 1 \\ & x_i \in [0, 1], \quad i = 1, \dots, N \end{aligned} \tag{1}$$

To avoid the non-differentiability of the *max* operator in (1), we reformulate the optimization problem according to Taylor and Bendsøe [33]. We introduce a positive auxiliary optimization variable, denoted $x_0 \in [0, \infty)$, which serves both as the cost function and as a uniform bound for all devices' latency. The new optimization problem reads:

$$\begin{aligned} & \min_{x, x_0} x_0 \quad (2.1) \\ \text{s.t.} \quad & C_i x_i \le x_0, \quad i = 1, \dots, N \quad (2.2) \\ & E_i(x_i, y_i) \le E^i_{available}, \quad i = 1, \dots, N \quad (2.3) \\ & \textstyle\sum_{i=1}^N x_i = 1 \quad (2.4) \\ & x_i \in [0, 1], \quad i = 1, \dots, N \quad (2.5) \\ & x_0 \ge 0 \quad (2.6), \end{aligned} \tag{2}$$

which is a linear programming (LP) problem and can be solved via standard dedicated algorithms (SIMPLEX, etc.) [34–36]. In more detail, the objective function (2.1) contains merely the auxiliary variable x_0. The additional constraints due to the problem reformulation are shown in (2.2). Then, the energy constraints, ensuring that each device disposes the required energy to perform the attributed tasks, are shown in (2.3). Eq. (2.4) imposes the sum of portions $\{x_i\}_{i=1,\dots,N}$ of B sent to the end device to be equal to 1. Finally, Eqs. (2.5) and (2.6) are bound constraints for the optimization variables.

Table 1
Key parameters and variables.

Description	Parameters/Variables
x_i	Percentage of the total number of images for every device i.
B	Batch of images for processing (Batch step).
T^i_{Tx}	Time to send a single image to device i.
$T^i_{\mathrm{dl}}(y_i)$	Time for executing the object detector on a transmitted image on device i, using the neural network y_i.

Our optimization method is an adaptive offloading technique that monitors the device and network characteristics and optimally distributes tasks among devices based on the current condition of the system.

Remark. In the above formulation for the latency, a major assumption is made. The network characteristics (T^i_{Tx}) are assumed to remain constant until the end of the task execution process. This reasonable assumption simplifies significantly the formulation and the solution of the optimization problem. Our experimental results, presented in the sequel, validate that this does not affect significantly the expected performance, as long as the network speed fluctuates within some expected interval.

3.2. Optimal task distribution and neural network selection

In Section 3.1, the neural network index y_i for each device is considered to be fixed, i.e., each device implements a predetermined object detector. In this Section, we add a discrete optimization variable, denoted as $y = [y_1, \dots, y_N]$, that represents the neural network selection used for the object detection at each end device.

In more detail, this variable represents the object detector model files which have different sizes and are located on every end device. The edge server in this case not only provides the splitting of the images but further decides for every end device which model it should use in order to optimize the overall performance.

For problem (2), which does not include a predefined object detection accuracy constraint the optimal choice of neural network is trivial, since the fastest neural network is also the less energy consuming and constitutes evidently the optimal choice. However, this choice produces the worst possible mean accuracy, which is also a critical performance parameter to consider. To obtain a meaningful problem, we define the normalized mean accuracy of the whole object detection process as:

$$A(x, y) = \frac{mAP(x, y) - mAP_{min}}{mAP_{max} - mAP_{min}}, \tag{3}$$

where:

$$mAP(x, y) = \sum_i mAP(y_i) x_i \tag{4}$$

and mAP_{min}, mAP_{max} represent the minimum and maximum mean Average Precision possible, i.e., the average precision provided by the smallest and largest neural network correspondingly. Then, we add an accuracy constraint stating that the minimum normalized accuracy achieved in the processing procedure must be greater than a threshold value defined by the user. The new optimization problem reads:

$$\begin{aligned} & \min_{x, x_0, y} x_0 \\ \text{s.t.} \quad & C_i x_i \le x_0, \quad i = 1, \dots, N \\ & E_i(x_i, y_i) \le E^i_{available}, \quad i = 1, \dots, N \\ & A(x, y) \ge A_{min}, \quad A_{min} \in [0, 1] \\ & \textstyle\sum_{i=1}^N x_i = 1 \\ & x_i \in [0, 1], \quad i = 1, \dots, N \\ & x_0 \ge 0 \end{aligned} \tag{5}$$

3.3. Solution of the mixed integer non-linear optimization problem

The optimization problem (5) is a MINLP which is not suited for gradient-based algorithms. Treating the discrete variables as continuous and then rounding the optimized values to the closest integers does not guarantee the optimality nor the satisfaction of the constraints.

An alternative option consists in adopting stochastic algorithms, which can be adjusted to treat both discrete and continuous variables. However, several limitations emerge under this perspective. First, the computational complexity increases exponentially with the number of discrete optimization variables and it is even worse for continuous variables. Second, handling several constraints in stochastic algorithms is tricky and it can lead to poor convergence. Consequently, since the speed of the algorithm is a crucial parameter for the practical applicability of our method, using purely stochastic algorithms is not relevant.

To circumvent the aforementioned inconveniences we propose a mixture of a stochastic algorithm and linear programming, that is able to handle problem (5) efficiently. We exploit the fact that, for a fixed choice of neural network indices y, problem (5) is a linear programming problem in low dimension and can be solved rapidly using dedicated algorithms. Therefore, we delegate to the stochastic algorithm only the search of the optimal y, while all the rest information (optimal task distribution, constraints satisfaction) derives from solving, for fixed y, the following linear programming problem:

$$\begin{aligned} &\min_{x,x_0} x_0 \\ \text{s.t.}\quad & C_i x_i \le x_0, \quad i=1,\dots,N \\ & E_i(x_i,y_i) \le E^i_{available}, \quad i=1,\dots,N \\ & A(x,y) \ge A_{min}, \quad A_{min}\in[0,1] \\ & \textstyle\sum_{i=1}^n x_i = 1 \\ & x_i \in [0,1], \quad i=1,\dots,N \\ & x_0 \ge 0 \end{aligned} \tag{6}$$

To the best of our knowledge, this approach is novel in literature, at least in the framework of Reverse Task Offloading.

For the numerical implementation of our approach, we have used a standard “$\mu+\lambda$” Genetic Algorithm (GA) [37–39]. GA is a population-based algorithm, where a set of candidate solutions (population) evolves during the iterations of the algorithm (generations) until a termination criterion is satisfied. The population evolution is principally determined by two nature-inspired operations: recombination and mutation. The recombination operation combines randomly chosen members of the current population (parents) to create ancestors for the next generation (offsprings), while mutation introduces random changes in a small portion of the ancestors. For the next generation, the best-performing members among parents and children are chosen, such that the total population size remains constant. Iterating this procedure is expected to overwrite characteristics of dominant solutions over the population, such that after some point no further improvement is achieved, indicating algorithmic convergence.

The pseudo-code of the EAI-ARO algorithm is shown in Algorithm 1. In line 1, the initial population $B_p^{(0)}$ is created. It is composed of μ y vectors containing combinations of the neural network indices. Lines 2–13 contain the optimization loop that iterates until some convergence criterion is satisfied. In every iteration, steps 3–12 are repeated. More precisely, lines 3–9 describe the creation of the offsprings population. In line 4, two members of $B_p^{(g)}$ are randomly chosen to become parents P_i for the new offspring. Applying the recombination (via crossover) operator on P_i we create the offspring O_i in line 5. Then, for a small percentage of offsprings, randomly chosen, we apply mutation in line 6 to obtain the final offspring $\bar{O}_i$. In order to compute the fitness function for $\bar{O}_i$, we solve the LP problem (6) in line 7, then compute the latency in line 8. Once the loop has been completed, we gather all offsprings and their corresponding performance in the offspring population $B_o^{(g)}$ in line 10. In line 11, we apply the so-called “$\mu+\lambda$” selection operator to chose the best μ members among the parents and ancestors. In line 12, we update the generation index before starting the new iteration.

Table 2
Genetic Algorithm parameters.

Optimization parameters	Values
Population size	100
Number of parents (μ)	50
Number of children (λ)	50
Number of mutants	10
max number of iterations	500
max number of successive refused iterations	100

Algorithm 1 Mixed GA-LP algorithm (EAI-ARO)

```
1:  $g$:=0, create initial population $B_p^{(0)}$
2:  loop:
3:      for $i := 1$ to $\lambda$
4:          random choice of parents $P_i$
5:          $O_i$ = recombination($P_i$)
6:          $\bar{O}_i$ = mutation($O_i$)
7:          solve LP problem (6) for $\bar{O}_i$
8:          calculate fitness $F_i$ for $\bar{O}_i$
9:      end
10:     $B_o^{(g)} = \{\bar{O}_i, F_i\}_{i=1,\dots,\lambda}$
11:     $(\mu+\lambda)$: $B_p^{(g)}$ = best($B_p^{(g)}, B_o^{(g)}, \mu$);
12:     $g$:=$g$+1
13: until convergence criterion;
```

After extensive numerical experiments, our choice of parameters for Algorithm 1 appears in Table 2. Concerning convergence criteria, we terminate our algorithm after a maximum number of successive refused iterations, i.e., where the objective function does not reduce, or after some total number of iterations.

3.4. Greedy reverse task offloading algorithm

We follow the methodology adopted in [18] and compare our proposed solution with a greedy algorithm, adjusted to fit the setup of our scenario. The used greedy algorithm assumes a fixed choice of neural network indices y and determines the task distribution variables x_i based on some notion of optimality [40]. It is composed of two steps detailed in the sequel.

First, the task distributions x_i are determined to be optimal for the latency objective, neglecting the accuracy and energy constraints, i.e., they solve the optimization problem:

$$\begin{aligned} &\min_x \max_i C_i x_i \\ s.t.\quad & \textstyle\sum_{i=1}^N x_i = 1 \end{aligned} \tag{7}$$

The analytical solution of (7) reads (see Appendix):

$$x_i = \frac{C}{C_i}, \quad \text{where:} \quad C = \frac{1}{\sum_{i=1}^n \frac{1}{C_i}}. \tag{8}$$

Then, we apply a heuristic step to handle the accuracy and energy constraints. In case the initial task distribution violates the accuracy constraint, the algorithm gradually redistributes tasks by assigning fewer tasks to devices with lower accuracy and more tasks to devices with higher accuracy, until the accuracy constraint is satisfied. Finally, the algorithm ensures that the energy constraint for each device is respected by setting an upper bound for each variable x_i as: $x_i^{max} = \frac{E^i_{available}}{BE(y_i)}$, which is the highest task distribution respecting the energy constraint for each device.

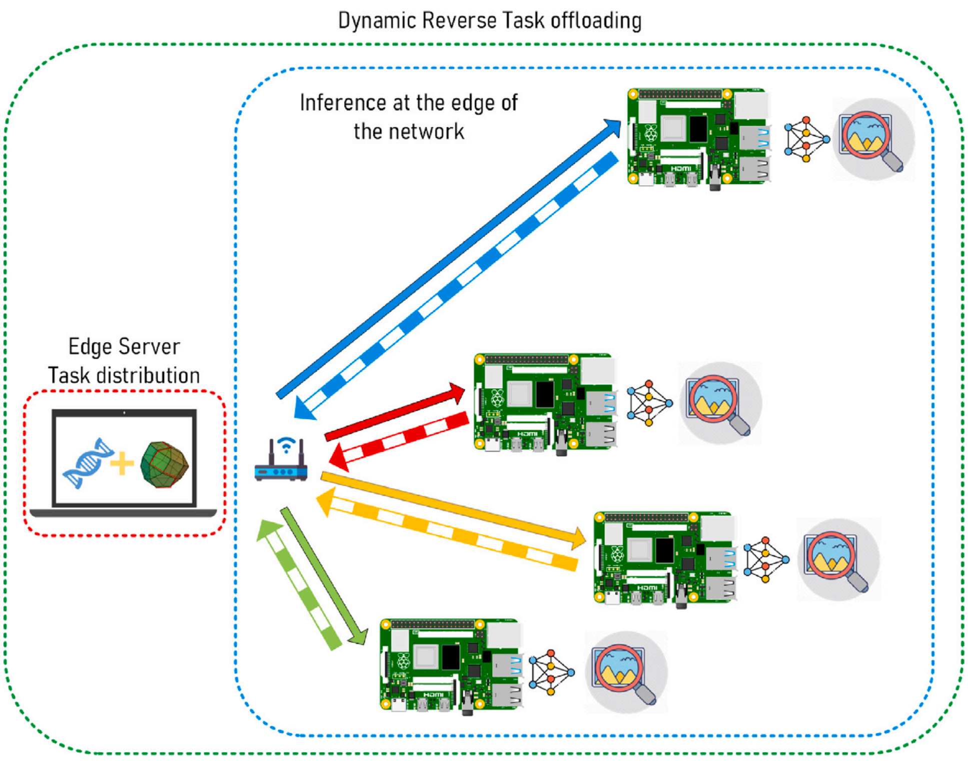

Dynamic Reverse Task offloading
Inference at the edge of the network
Edge Server Task distribution

Fig. 2. Testbed topology.

4. Results

In this section, we present both experimental and numerical results using the proposed adaptive offloading approach. Numerical results derive from solving numerically the optimization problem at stake using artificial network values for each link between the edge server and end device. On the other hand, experimental results correspond to the findings of our method when implemented in our testbed under real-world network conditions.

In the first step, we present our testbed and compare the EAI-ARO with the fixed offloading baseline and greedy algorithm. The fixed offloading baseline algorithm offloads a predetermined percentage or portion of the task to each end device, regardless of its current network condition. More specifically, we evenly distribute tasks among devices in our testbed. Then, we examine how the expected gain in time varies with the characteristics of the network. Finally, we examine the efficiency of our approach for large-scale systems and propose a clustering technique that maximizes the gain in latency.

4.1. Testbed

We present an edge computing setup that consists of four end devices, an edge server and a Wi-Fi access point, as shown in Fig. 2.

The end devices correspond to Raspberry Pi's 4 Model B with 4 GB of RAM along with the corresponding wireless network connections. They perform object detection tasks using YOLOv8, a deep-learning object detector. They use YOLOv8, a deep-learning object detector, to carry out object detection tasks. In more detail, the input for the YOLOv8 model corresponds to a $n \times n \times 3$ array of image pixels, each of which can be represented as an integer or floating-point value. After downsampling this input array, a grid of cells is produced that suggests labels and bounding boxes for the objects in the dataset. As a result, a set of bounding boxes for the identified objects is produced, completed with labels and confidence values. Each end device may dispose several

Table 3
Characteristics of object detection models.

y_i index	Input image size	$mAP(y_i)$	$E(y_i)$ (W/image)
1	224 × 224 × 3	0.325	0.6
2	320 × 320 × 3	0.415	1.22
3	480 × 480 × 3	0.481	2.16
4	640 × 640 × 3	0.531	4.8

YOLOv8 models, implemented with different input image sizes, energy consumption values and mAP values, as shown in Table 3.

The edge server, which is essentially a laptop powered by an i7-12700H CPU processor equipped with 32 GB of RAM, distributes batches of images (picked from the COCO dataset) for object detection to the four end devices simultaneously. The results, which include labels for every processed image and the corresponding bounding boxes, are then transmitted back by the end devices, the device that finishes the processing of the images last is considered as the latency.

The total batch corresponds to 20 000 images of about 150 KB and every batch step equals 2000 images, i.e., the process completes after 10 batch steps. At the end of every task execution process, an optimization problem is solved taking into account the current network state (see Fig. 3), i.e., the inherent dynamic problem is approximated as a series of static problems. To better explain the imbalance between devices, each device in the network due to distance to the access point has a different range of network speed values. Specifically, those located closer to the access point benefit from a stronger connection and consequently higher network speeds, whereas devices situated further away experience weaker connections and reduced network speeds. The whole distribution mechanism is coded in Python.

4.2. Optimal task distribution

In our first example, each end device i is attributed the corresponding object detector with index i, i.e., $y = [1, 2, 3, 4]$ is considered

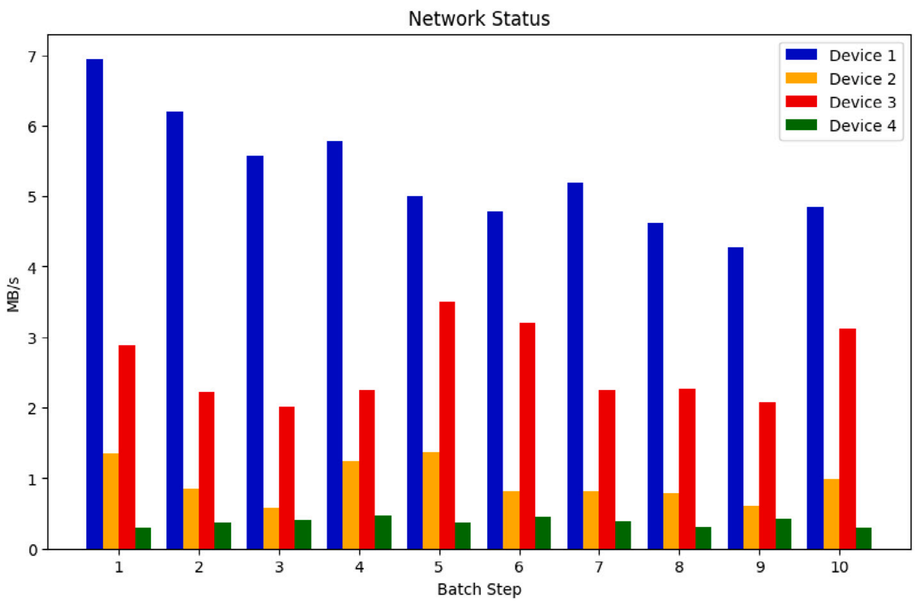


Fig. 3. Network status at every batch step.

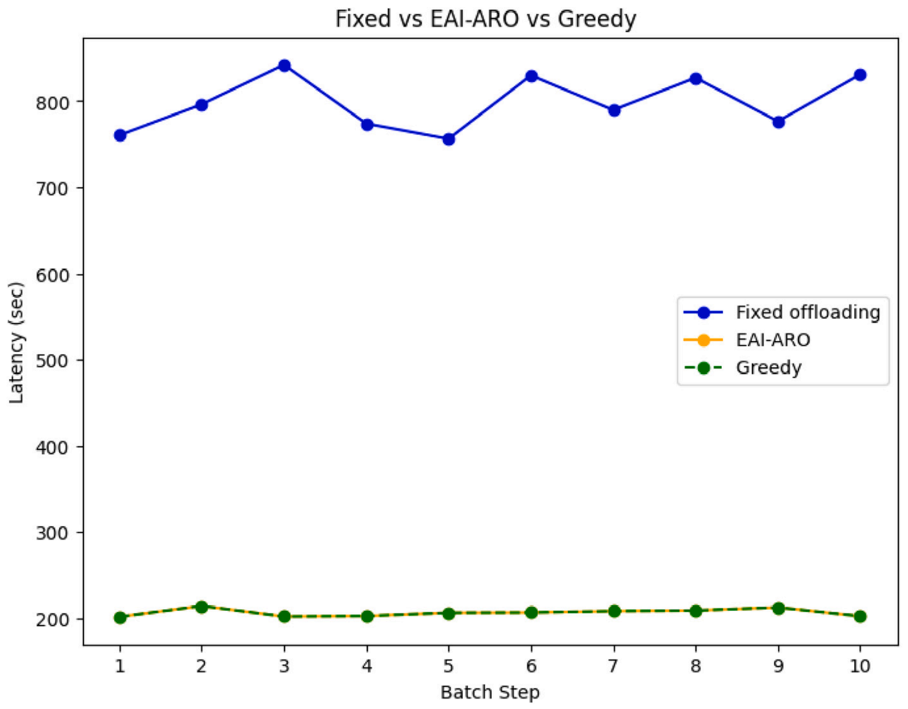


Fig. 4. Comparison of latency at each batch step between the fixed offloading, the greedy and the EAI-ARO algorithm.

as constant. Moreover, the available energy is assumed to be equal to 20 000 (mAh) for all end devices. We search to optimize the task distribution between the devices solving the optimization problem (2) at every batch step for the testbed described in Section 4.1.

In Fig. 4, we plot the numerical evolution of the latency at every batch step for the fixed offloading, the greedy, and the EAI-ARO algorithm. First, let us note that as long as the energy constraints remain inactive, both the used greedy and the EAI-ARO algorithms solve the optimization problem (2), i.e., the provided solutions are equivalent. Moreover, one can observe that their solution outperforms significantly the fixed task offloading at every batch step. In average, the gain in latency equals 73.5% for the numerical results and 72.3% for the experimental results (see Table 4).

In Fig. 5, we also provide the corresponding evolution of x_i, i.e., the task distribution for each device at every batch step. It is obvious that due to the fact that device 1 has nearly 50% better network connection on average compared to the other devices, this leads to receiving almost 56% on average more task load at each batch step.

The above gain highlights the importance of employing the EAI-ARO algorithm when handling time-sensitive applications. In fact, in case of a very bad connection between the server and some end device, which additionally may be provided with a slow object detection model, the fixed offloading algorithm may result in excessive processing time. This situation is avoided in the EAI-ARO approach by providing a sufficiently low task percentage to the corresponding device.

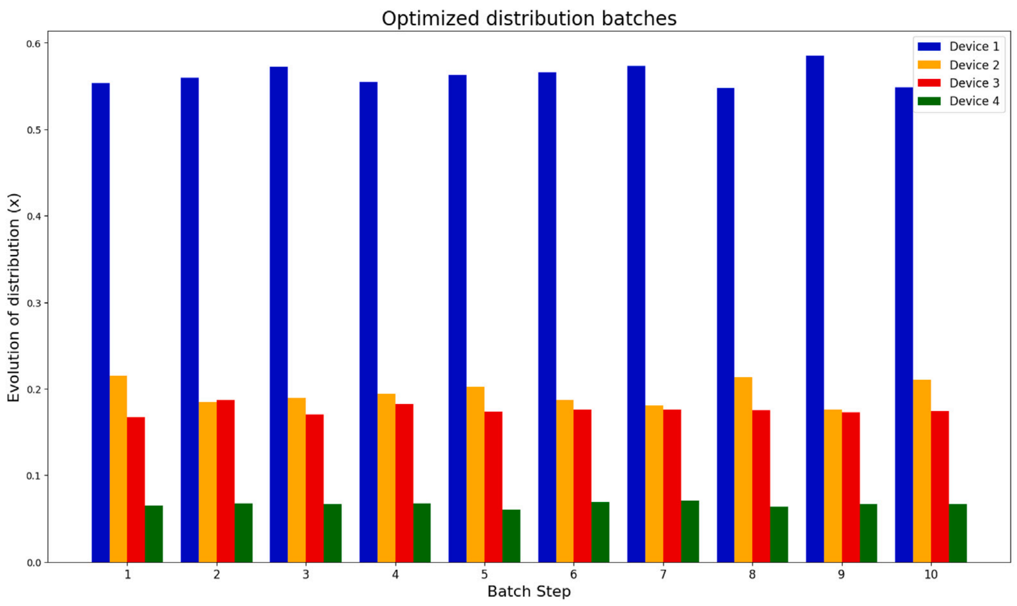


Fig. 5. Evolution of distributions at every batch step and device.

Table 4
Average latency for the results in Fig. 4.

Results	Fixed offloading (s)	Greedy (s)	EAI-ARO (s)	Gain (%)
Numerical	785	208	208	73.5
Experimental	779	215	215	72.3

4.3. *Optimal task distribution and neural network selection*

In the following example, beyond optimizing the task distribution, each end device is further provided with the complete set of the available object detectors and the selection at each batch step, using the EAI-ARO method, is determined by the solution of the optimization problem (5). We examine the gains of our proposed method for two different thresholds (A_{min}) of the accuracy constraint, which correspond to different fixed selections of neural networks for the greedy and the fixed offloading methods. More specifically, we examine the case of $y = [1, 1, 1, 1]$ and $y = [2, 2, 3, 4]$ that provide $A_{min} = 0$ and $A_{min} = 0.47$ correspondingly.

Once more, we test our approach both numerically and experimentally. The obtained results (see Table 5) show that the proposed approach provides significant gains in terms of latency, compared to the greedy and fixed task distribution method. In Fig. 6, we plot the evolution of the latency at every batch step for all methods. As expected, the gain depends strongly on the specified accuracy threshold. In more detail, setting $A_{min} = 0$ turns the optimal solution of y to become trivial and equal to $[1, 1, 1, 1]$, i.e., all three methods use the same neural networks. As we mentioned before if all devices dispose of the needed energy to perform the attributed tasks, the EAI-ARO and greedy approach results coincide. On the other hand, when the accuracy threshold is set to $A_{min} = 0.47$, the optimal y choice is non-trivial, therefore there is margin for further gain. This is reflected in the results in Table 5. As expected the gain of the EAI-ARO method with respect to the greedy approach is smaller compared to the fixed task offloading algorithm. Finally, in Fig. 6(b) one may observe a sudden increase in latency for all methods at different batch steps. This increase corresponds to instances when some energy constraint becomes active, i.e., some device runs out of battery. It is clear from the latency evolution that the EAI-ARO algorithm is more robust since it is less affected compared to the other two methods.

4.4. *Impact of network characteristics*

In this section, we test numerically the efficiency of our approach under different network conditions. In more detail, instead of a network that varies for each connection between the server and the end devices, we consider uniform network conditions. Three different scenarios are examined, shown in Table 6.

For each scenario, we first optimize the task distribution considering fixed the Neural Network selection (see Section 4.2). The results are shown in Table 7. As expected, the gain is reduced for slower networks, since the object detection task becomes more significant than the image transmission for the latency.

Then, we also provide the possibility to optimize the choice of the Neural Network used. The results appear in Table 8. One may verify therein the expected tendencies concerning the network speed, as well as the accuracy limit.

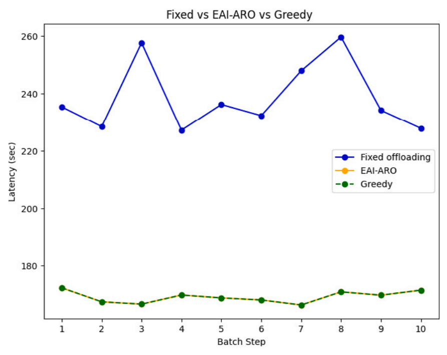
4.5. *Large-scale testbed*

In the previous sections, we validated the efficiency of our proposal using a testbed comprised of four end devices. In order for the method (EAI-ARO) to be applicable for real-life applications, we need to further examine the efficiency of the approach in large-scale testbeds. This is achieved in a numerical framework described in the sequel of this section.

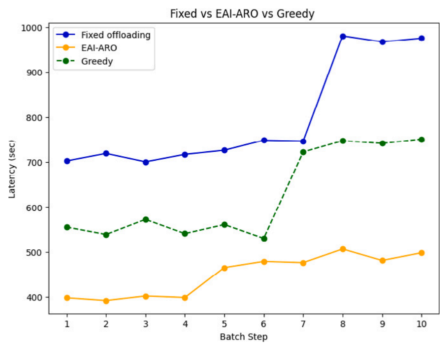
4.5.1. *Clustering technique*

For small testbeds, as the one in Section 4.1, the time for the resolution of the MINLP problem can be assumed negligible compared to the processing time of the task. However, as the number of end devices increases, the cost of the Genetic Algorithm is expected to increase exponentially, in accordance to the possible combinations of indices composing the y vector. It shall be expected that the total time (processing + solution of MINLP problem) increases after some limit value of end devices, which also depends on the characteristics of the particular application (network status, batch step, testbed, etc.). When the number of end devices exceeds this limit value, devices shall be clustered to maximize the possible gain. For each cluster, a separate MINLP problem is solved.

In our numerical test for detecting the optimal cluster size, we consider a testbed composed of 128 end devices. For the network status,



(a) $A_{min} = 0$.



(b) $A_{min} = 0.47$.

Fig. 6. Evolution of latency for different accuracy thresholds for all task distribution methods.

Table 5
Average latency and gain for the results in Fig. 6.

Experiments	Accuracy threshold	Fixed offloading (s)	Greedy (s)	EAI-ARO (s)	Gain (Fixed) (%)	Gain (Greedy) (%)
Numerical	0	238.6	238.6	169.1	29.1	0
Experimental	0	268	268	200	25.3	0
Numerical	0.47	707	555.7	397.7	43.7	28.4
Experimental	0.47	740.1	592	438.3	40	28.9

Table 6
Uniform network scenario characteristics.

Network average speed (MB/s)	Network speed interval (MB/s)
5.9	4.8–7
3.0	2.2–3.8
0.35	0.2–0.5

Table 7
Average latency for the uniform network scenario.

Inteval (MB/s)	Fixed offloading (s)	EAI-ARO (s)	Gain (%)
4.8–7	610	178	70.1
2.2–3.8	745	285.2	65.3
0.2–0.5	824	400	51.4

we consider random values varying between the limit measurements in Fig. 3. Provided the stochastic nature of our algorithm, to achieve reliable results, we execute the process 200 times for each cluster size in the set $\{4, 8, 16, 32, 64, 128\}$.

The outcomes of the calibration procedure are illustrated in Fig. 7. The cluster latency reduces as the number of devices in each cluster increases. This is expected since the degrees of freedom in the MINLP problem increases and the optimization algorithm (EAI-ARO) can achieve more efficient distributions and y combinations. However, gains are not significant for more than 16 end devices. On the contrary, the curve representing the total time presents a minimum value that is achieved for 32 end devices. This value is considered as the optimal configuration for the numerical tests in Section 4.5.2.

4.5.2. Numerical validation

In this section, we create numerically testbeds in our heterogeneous scenario for 32 end devices which is the optimal configuration according to Section 4.5.1 and compare our EAI-ARO distribution proposal with the greedy and the uniform task offloading algorithm. The creation of the testbeds is performed by replicating the characteristics of the architecture described in Section 4.1. As in Section 4.5.1, the artificial network status is constructed by considering random values varying between the limit measurements in Fig. 3.

First, we examine the optimal task distribution for fixed object detection models, solving the optimization problem (2) for different testbeds. The results appear in Table 9, validating a significant gain of 57.2% compared to the fixed task offloading algorithm. We highlight here that the solution of the optimization problem (2), which is a Linear Programming problem, requires negligible time to reach the global optimum and it can be extended without problem for much larger testbeds.

Then, we provide the possibility for every device to implement any of the Neural Networks described in Table 3, solving the MINLP problem (5) for different testbeds and two accuracy limits. We highlight that for more than 32 end devices, which is the optimal cluster size found in Section 4.5.1, the clustering technique is applied. For the trivial y selection corresponding to $A_{min} = 0$ the results are shown in Table 10. One can verify that our proposed method provides a gain in terms of latency of 39.2% compared to the fixed task offloading algorithm. For the non-trivial case of $A_{min} = 0.58$ the results are also shown in Table 10. The gain in latency increases to 40% when compared to the fixed algorithm and 13.5% compared to the greedy algorithm, which verifies once more that the additional optimization variables corresponding to the object detection models enhance the performance of the system without reducing the mean accuracy.

Reverse offloading can be beneficial in scenarios where the computational resources of edge devices are either idle or more effective for specific tasks than centralized cloud servers. In such cases, the adoption of reverse offloading is recommended. This method minimizes latency and prevents bottlenecks by reducing the requirement for continuous data transmission to the cloud, which is especially helpful in situations where network bandwidth is limited. To achieve faster response times, reverse offloading is particularly helpful for tasks requiring real-time processing or when the edge devices are located closer to the data source. Furthermore, by utilizing local processing power, this technique can increase energy efficiency by lowering the total energy consumption related to cloud computing and data transfer. Systems that use reverse offloading instead of the traditional task offloading in some circumstances can achieve better load balancing, improve reliability through localized processing, and provide more immediate feedback.

5. Conclusion

We developed an effective optimization-based distribution mechanism for reverse task execution of AI applications in resource-constrained IoT networks. The motivation behind this work derives from the rapid increase of computational power and the improved

Table 8
Average latency for the uniform network scenario and different accuracy thresholds.

Interval (MB/s)	A_{min}	Fixed offloading (s)	Greedy (s)	EAI-ARO (s)	Gain fixed (%)	Gain greedy (%)
4.8–7	0	98	85	85	13.2	0
4.8–7	0.58	620.1	445.5	378.2	28.1	15.1
2.2–3.8	0	113	105	105	7	0
2.2–3.8	0.58	634.1	468.5	398.3	26.1	14.9
0.2–0.5	0	309	290	290	6.1	0
0.2–0.5	0.58	780.2	579.8	502.3	25.7	13.2

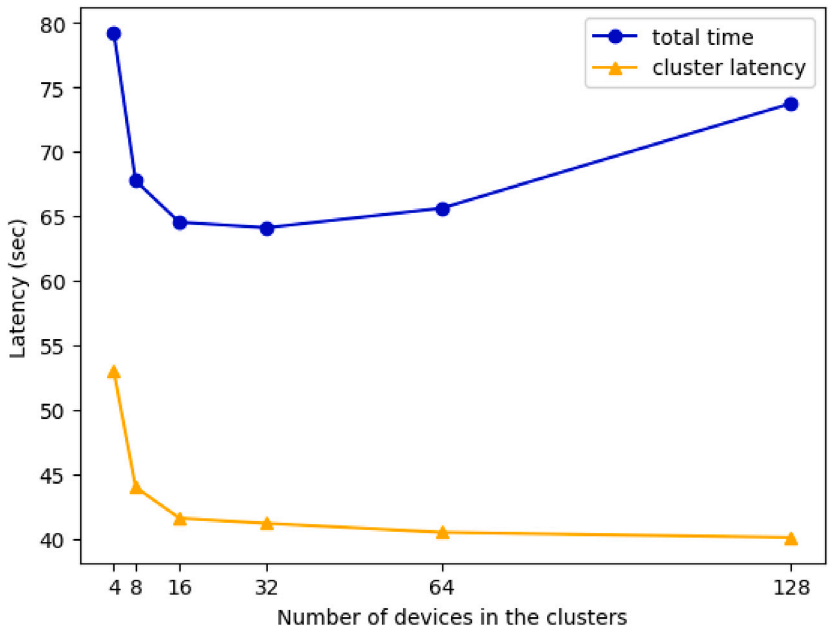


Fig. 7. Calibration of the optimal cluster size.

Table 9
Average latency for 32 devices and fixed models.

Fixed offloading (s)	Greedy (s)	EAI-ARO (s)	Gain (%)
101	43.2	43.2	57.2

storage capacity of IoT devices, resulting in new capabilities for IoT applications. The distribution mechanism called EAI-ARO (Edge AI-Adaptive Reverse offloading) is designed to determine the optimal task-splitting strategy in a systematic fashion, solving corresponding optimization problems. We propose a novel approach that combines different algorithms to solve the formulated MINLP-type optimization problem. We fully assessed the performance of our proposal, utilizing it in an object detection application within a Raspberry Pi testbed. Furthermore, we executed a numerical validation in a large-scale testbed to determine the scalability of our method for real-world systems and suggested a strategy based on clustering of end devices. The results unequivocally demonstrate that our mechanism surpasses the fixed task offloading baseline and the greedy algorithm among end devices and increases the ability to handle AI workloads improving significantly the system's overall performance. Our findings open up fascinating new directions for future research. These include the design of task distribution mechanisms based on multi-objective optimization problems, accounting both for latency minimization, energy efficiency, and accuracy maximization, which can be implemented and used for real-time applications without the use of cloud resources. Additionally, a predictive offloading method, where decisions are based on machine learning models predicting future network conditions, could further enhance the efficiency of our reverse offloading computing systems.

CRediT authorship contribution statement

Petros Amanatidis: Writing – original draft, Visualization, Software, Methodology, Formal analysis. **Dimitris Karampatzakis:** Writing – review & editing, Supervision, Project administration, Data curation, Conceptualization. **Georgios Michailidis:** Validation, Methodology, Investigation, Formal analysis. **Thomas Lagkas:** Writing – review & editing, Supervision, Project administration, Funding acquisition. **George Iosifidis:** Writing – review & editing, Supervision, Methodology, Formal analysis.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: THOMAS LAGKAS reports financial support was provided by Horizon Europe. THOMAS LAGKAS reports article publishing charges was provided by HEAL-LINK. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070181.

The publication of the article in OA mode was financially supported in part by HEAL-Link.

Table 10
Average latency for 32 devices and different accuracy thresholds.

Accuracy threshold	Fixed offloading (s)	Greedy (s)	EAI-ARO (s)	Gain fixed (%)	Gain greedy (%)
0	39.7	24.1	24.1	39.2	0
0.58	92	68.5	55.2	40	13.5

Appendix. Proof of Eq. (8)

We reformulate the optimization problem (7) as:

$$\begin{aligned} &\min_{x,x_0} x_0 \\ s.t.\quad &C_i x_i \le x_0, \quad i = 1,\dots,N \\ &\textstyle\sum_{i=1}^{N} x_i = 1 \end{aligned} \tag{A.1}$$

and define the Lagrangian function:

$$L(x,x_0,\lambda,\mu) = x_0 + \sum_{i=1}^{N} \lambda_i (C_i x_i - x_0) + \mu(\sum_{i=1}^{N} x_i - 1), \tag{A.2}$$

where $\lambda = [\lambda_1,\dots,\lambda_N] \in R^N$ are positive lagrange multipliers for the inequality constraints and $\mu \in R$ is a lagrange multiplier for the equality constraint. The KKT optimality conditions are revealed by setting the partial derivatives of the L equal to zero:

$$\frac{\partial L(x,x_0,\lambda,\mu)}{\partial x_i} = 0 \Rightarrow \lambda_i C_i + \mu = 0 \tag{A.3}$$

$$\frac{\partial L(x,x_0,\lambda,\mu)}{\partial x_0} = 0 \Rightarrow \sum_{i=1}^{N} \lambda_i = 1 \tag{A.4}$$

$$\frac{\partial L(x,x_0,\lambda,\mu)}{\partial \mu} = 0 \Rightarrow \sum_{i=1}^{N} x_i = 1 \tag{A.5}$$

If $\lambda_i = 0$ $i = 1,\dots,N$, from Eq. (A.3) we conclude that $\mu = 0$ and $\lambda_i = 0 \quad \forall i = 1,\dots,N$, which is impossible due to Eq. (A.4). Therefore $\lambda_i > 0$ and:

$$\frac{\partial L(x,x_0,\lambda,\mu)}{\partial \lambda_i} = 0 \Rightarrow C_i x_i = x_0 \Rightarrow x_i = \frac{x_0}{C_i} \tag{A.6}$$

Using Eqs. (A.5) and (A.6) we get:

$$\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} \frac{x_0}{C_i} = 1 \Rightarrow x_0 \sum_{i=1}^{N} \frac{1}{C_i} = 1 \Rightarrow x_0 = \frac{1}{\sum_{i=1}^{N} \frac{1}{C_i}} \tag{A.7}$$

Substituting Eq. (A.7) in (A.6) concludes the proof.

References

[1] F. Wang, M. Zhang, X. Wang, X. Ma, J. Liu, Deep learning for edge computing applications: A state-of-the-art survey, IEEE Access 8 (2020) 58322–58336.
[2] Y. Sun, H. Ochiai, H. Esaki, Decentralized deep learning for multi-access edge computing: A survey on communication efficiency and trustworthiness, IEEE Trans. Artif. Intell. 3 (6) (2021) 963–972.
[3] C. Campolo, A. Iera, A. Molinaro, Network for distributed intelligence: A survey and future perspectives, IEEE Access 11 (2023) 52840–52861, http://dx.doi.org/10.1109/ACCESS.2023.3280411.
[4] D.D. Sivaganesan, Design and development ai-enabled edge computing for intelligent-iot applications, J. Trends Comput. Sci. Smart Technol. 1 (2) (2019) 84–94.
[5] D. Rosendo, A. Costan, P. Valduriez, G. Antoniu, Distributed intelligence on the edge-to-cloud continuum: A systematic literature review, J. Parallel Distrib. Comput. 166 (2022) 71–94.
[6] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, A.Y. Zomaya, Edge intelligence: The confluence of edge computing and artificial intelligence, IEEE Internet Things J. 7 (8) (2020) 7457–7469, http://dx.doi.org/10.1109/JIOT.2020.2984887.
[7] P. Wang, C. Yao, Z. Zheng, G. Sun, L. Song, Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems, IEEE Internet Things J. 6 (2) (2019) 2872–2884, http://dx.doi.org/10.1109/JIOT.2018.2876198.
[8] J. Zhao, Q. Li, Y. Gong, K. Zhang, Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks, IEEE Trans. Veh. Technol. 68 (8) (2019) 7944–7956, http://dx.doi.org/10.1109/TVT.2019.2917890.
[9] K. Sadatdiynov, L. Cui, L. Zhang, J.Z. Huang, S. Salloum, M.S. Mahmud, A review of optimization methods for computation offloading in edge computing networks, Digit. Commun. Netw. (2022).
[10] C.-F. Liu, M. Bennis, M. Debbah, H.V. Poor, Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing, IEEE Trans. Commun. 67 (6) (2019) 4132–4150, http://dx.doi.org/10.1109/TCOMM.2019.2898573.
[11] W. Feng, N. Zhang, S. Li, S. Lin, R. Ning, S. Yang, Y. Gao, Latency minimization of reverse offloading in vehicular edge computing, IEEE Trans. Veh. Technol. 71 (5) (2022) 5343–5357, http://dx.doi.org/10.1109/TVT.2022.3151806.
[12] A. Ali, M.M. Iqbal, H. Jamil, F. Qayyum, S. Jabbar, O. Cheikhrouhou, M. Baz, F. Jamil, An efficient dynamic-decision based task scheduler for task offloading optimization and energy management in mobile cloud computing, Sensors 21 (13) (2021) 4527.
[13] Y. Chen, F. Zhao, Y. Lu, X. Chen, Dynamic task offloading for mobile edge computing with hybrid energy supply, Tsinghua Sci. Technol. 28 (3) (2022) 421–432.
[14] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, J. Rodriguez, Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach, IEEE Trans. Veh. Technol. 68 (4) (2019) 3113–3125, http://dx.doi.org/10.1109/TVT.2019.2894851.
[15] A. Galanopoulos, T. Salonidis, G. Iosifidis, Cooperative edge computing of data analytics for the internet of things, IEEE Trans. Cognit. Commun. Netw. 6 (4) (2020) 1166–1179.
[16] A. Galanopoulos, G. Iosifidis, T. Salonidis, D.J. Leith, Selective edge computing for mobile analytics, IEEE Trans. Netw. Serv. Manag. 19 (3) (2022) 3090–3104, http://dx.doi.org/10.1109/TNSM.2022.3174776.
[17] C. Feng, P. Han, X. Zhang, B. Yang, Y. Liu, L. Guo, Computation offloading in mobile edge computing networks: A survey, J. Netw. Comput. Appl. 202 (2022) 103366, http://dx.doi.org/10.1016/j.jnca.2022.103366, URL: https://www.sciencedirect.com/science/article/pii/S1084804522000327.
[18] V. Cozzolino, L. Tonetto, N. Mohan, A.Y. Ding, J. Ott, Nimbus: Towards latency-energy efficient task offloading for ar services, IEEE Trans. Cloud Comput. 11 (2) (2022) 1530–1545.
[19] Y. Gong, K. Bian, F. Hao, Y. Sun, Y. Wu, Dependent tasks offloading in mobile edge computing: A multi-objective evolutionary optimization strategy, Future Gener. Comput. Syst. 148 (2023) 314–325.
[20] M. Maray, E. Mustafa, J. Shuja, M. Bilal, Dependent task offloading with deadline-aware scheduling in mobile edge networks, Internet of Things 23 (2023) 100868.
[21] T.X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, IEEE Trans. Veh. Technol. 68 (1) (2019) 856–868, http://dx.doi.org/10.1109/TVT.2018.2881191.
[22] T. Lin, M. Maire, S.J. Belongie, L.D. Bourdev, R.B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, C.L. Zitnick, Microsoft COCO: common objects in context, 2014, CoRR abs/1405.0312. arXiv:1405.0312. URL: http://arxiv.org/abs/1405.0312.
[23] Q. You, B. Tang, Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things, J. Cloud Comput. 10 (2021) 1–11.
[24] H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2022) 1517–1530, http://dx.doi.org/10.1109/JIOT.2021.3091142.
[25] Z. Ali, Z.H. Abbas, G. Abbas, A. Numani, M. Bilal, Smart computational offloading for mobile edge computing in next-generation internet of things networks, Comput. Netw. 198 (2021) 108356, http://dx.doi.org/10.1016/j.comnet.2021.108356, URL: https://www.sciencedirect.com/science/article/pii/S1389128621003467.
[26] Z. Ning, Y. Yang, X. Wang, L. Guo, X. Gao, S. Guo, G. Wang, Dynamic computation offloading and server deployment for UAV-enabled multi-access edge computing, IEEE Trans. Mob. Comput. 22 (5) (2023) 2628–2644, http://dx.doi.org/10.1109/TMC.2021.3129785.
[27] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, H. Jin, Online collaborative data caching in edge computing, IEEE Trans. Parallel Distrib. Syst. 32 (2) (2021) 281–294, http://dx.doi.org/10.1109/TPDS.2020.3016344.
[28] G. Pan, H. Zhang, S. Xu, S. Zhang, X. Chen, Joint optimization of video-based AI inference tasks in MEC-assisted augmented reality systems, IEEE Trans. Cognit. Commun. Netw. 9 (2) (2023) 479–493, http://dx.doi.org/10.1109/TCCN.2023.3235773.
[29] J. Zheng, K. Li, N. Mhaisen, W. Ni, E. Tovar, M. Guizani, Federated learning for online resource allocation in mobile edge computing: A deep reinforcement learning approach, in: 2023 IEEE Wireless Communications and Networking Conference, WCNC, 2023, pp. 1–6, http://dx.doi.org/10.1109/WCNC55385.2023.10118940.
[30] Y. Kim, C. Song, H. Han, H. Jung, S. Kang, Collaborative task scheduling for IoT-assisted edge computing, IEEE Access 8 (2020) 216593–216606, http://dx.doi.org/10.1109/ACCESS.2020.3041872.

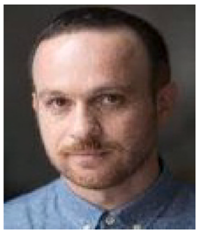
[31] J. Feng, W. Feng, S. Lin, Reverse computing offloading for enhanced computing capacity in cooperative vehicle infrastructure system, in: 2021 IEEE International Intelligent Transportation Systems Conference, ITSC, 2021, pp. 1011–1016, http://dx.doi.org/10.1109/ITSC48978.2021.9564763.
[32] G. Jocher, A. Chaurasia, J. Qiu, YOLO by ultralytics. 2023, 2023, URL: https://github.com/ultralytics/ultralytics.
[33] J. Taylor, M.P. Bendsøe, An interpretation for min-max structural design problems including a method for relaxing constraints, Int. J. Solids Struct. 20 (4) (1984) 301–314.
[34] D. Bertsimas, J.N. Tsitsiklis, Introduction to Linear Optimization, vol. 6, Athena Scientific Belmont, MA, 1997.
[35] G.B. Dantzig, M.N. Thapa, Linear Programming: Theory and Extensions, vol. 2, Springer, 2003.
[36] J.A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313.
[37] J.H. Holland, Genetic algorithms, Sci. Amer. 267 (1) (1992) 66–73.
[38] M. Gen, R. Cheng, Genetic Algorithms and Engineering Optimization, vol. 7, John Wiley & Sons, 1999.
[39] S. Sivanandam, S. Deepa, S. Sivanandam, S. Deepa, Genetic algorithm optimization problems, in: Introduction to Genetic Algorithms, Springer, 2008, pp. 165–209.
[40] P.E. Black, Greedy algorithm, dictionary of algorithms and data structures, US Natl. Inst. Std. Tech. Rep. 88 (2012) 95.

Petros Amanatidis was born in 1994 in Hamburg, Germany. He received his B.Sc in Computer Science from the Department of Computer Science at the International Hellenic University, Kavala, Greece, in 2019. In 2021, he completed his postgraduate studies (Mphil) in Advanced Technologies in Informatics and Computers from the same department. He is currently a PhD Candidate at the Department of Informatics of the Democritus University of Thrace focusing on Edge Computing, more specifically on the methodologies for optimal data and resource management. In parallel, he works as Research Assistant in several EU-funded research projects.

Dimitris Karampatzakis received his Diploma degree in electronics and computer engineering from the Technical University of Crete, in 2003, and the Ph.D. degree from the University of Thessaly in 2009. He is an assistant professor at the Department of Informatics, Democritus University of Thrace. His current research interests include edge computing, internet of things applications and computer hardware. He has participated in national and european research projects, and publications in international scientific journals and conferences.

George Michailidis was born in Kavala in 1984. He studied Civil Engineering at Aristotle University of Thessaloniki. After completing his military service, he did postgraduate studies (MSc) in Applied Mathematics at the National Technical University of Athens and obtained his doctoral degree (PhD) from the Ecole Polytechnique of Paris. Currently, he serves as adjunct professor at the Department of Informatics, Democritus University of Thrace, Greece. He has authored numerous research papers and he has been a speaker at a number of scientific conferences worldwide. He has worked at Renault's research center (Renault Technocentre) and at two French universities (Ecole Polytechnique, Université Grenoble Alps). In 2017 he was hired as Software Developer at ANSYS Inc.

Thomas Lagkas received BSc degree (Hons.) and PhD degree in wireless networks from the Department of Computer Science, Aristotle University of Thessaloniki, in 2002 and 2006, respectively, MBA degree from Hellenic Open University, in 2012, and postgraduate certificate on teaching and learning from the University of Sheffield, in 2017. He is an assistant professor at the Department of Informatics, Democritus University of Thrace and Director of the Laboratory of Industrial and Educational Embedded Systems. He has been a scholar of the Aristotle University Research Committee, as well as a postdoctoral scholar of the National Scholarships Institute of Greece. His research interests are in the areas of IoT communications with more than 150 publications at a number of widely recognized international scientific journals and conferences. He is a fellow of the Higher Education Academy in U.K. Moreover, he actively participates in the preparation, management, and implementation of several EU-funded research projects.



George Iosifidis received the Diploma degree in electronics and telecommunications engineering from the Greek Air Force Academy, Athens, in 2000, and the Ph.D. degree from the University of Thessaly in 2012. He was an Assistant Professor at Trinity College Dublin from 2016 to 2020. He is an Associate Professor with the Delft University of Technology. His research interests lie in the broad area of network optimization and economics; more information can be found at www.FutureNetworksLab.net.