Journal of Network and Systems Management (2026) 34:54
https://doi.org/10.1007/s10922-025-10022-5

RASH: Resource Allocation for Smart Homes Considering Privacy Sensitivity of IoT Applications

Tina Rezaei[1] · Suzan Bayhan[1] · Andrea Continella[1] · Roland van Rijswijk-Deij[1]

Received: 28 May 2025 / Revised: 11 November 2025 / Accepted: 23 December 2025

Abstract
Smart home applications have increasingly adopted compute-intensive, machine-learning based techniques to improve their efficiency and prediction capabilities, e.g., for the comfort of the residents. While edge computing has been the preferred computing paradigm in smart homes due to the latency, data privacy, and resilience benefits, the limited computational and bandwidth resources of local edge devices, e.g., a home gateway, might hinder the timely completion of Internet-of-Things (IoT) tasks, necessitating computation offloading of some tasks to remote compute nodes. While ensuring timely completion of an application is essential in offloading decisions, offloading without consideration of data privacy poses privacy risks. To mitigate such consequences, this paper presents RASH, a resource allocation and offloading scheme that has three merits. *First*, different from prior studies, RASH assigns a privacy-sensitivity score to the tasks based on their data type and the location of the IoT devices to prioritize local execution of highly sensitive tasks when local resources are insufficient and offloading is inevitable. *Second*, to handle peak load scenarios, RASH incorporates a heuristic task postponement algorithm that effectively defers tasks to maintain the problem's feasibility and maximize resource utilization. *Third*, RASH runs periodically, dynamically adjusting resource allocation based on real-time computational demands and newly arriving *training* and *executing* tasks with varying time budgets and computational needs. Our evaluation results show that RASH effectively prioritizes the local execution of privacy-sensitive tasks with more than 90% local resource utilization.

Keywords Computation offloading · Smart home · IoT · Privacy

Our simulator can be found at https://github.com/Tina-Rezaei/RASH.

Extended author information available on the last page of the article

Published online: 18 February 2026

1 Introduction

Smart homes equipped with Internet-of-Things (IoT) devices capable of sensing and consequently actuating physical processes offer many benefits to the residents, such as more sustainable energy usage or smart health services [1, 2]. In addition to conventional smart devices only executing the commands received through a remote controller, there is a growing trend towards more sophisticated, machine learning-enabled applications [3], e.g., cameras performing image processing to detect individuals' faces, smart speakers analyzing the user's recorded voice, wearable sensing devices for healthcare in a residential environment to identify health conditions of the residents [1, 4]. However, these applications often exceed the computational capabilities of IoT devices, leading to reliance on cloud-based computation, which introduces potential privacy risks or unnecessary communication and data storage at the cloud. As a remedy, computing at the edge devices has attracted significant attention [5–8]. This approach not only decreases latency and traffic load but also enhances privacy as it enables computation at the proximity of the smart home devices, e.g., on a home router or a local hub located in the same residential space.

Despite the advantages of edge computing, equipping all smart homes with a large compute capacity might raise its own challenges, e.g., cost and sustainability. With growing computational demands, leveraging edge and cloud resources simultaneously becomes essential to meet the requirements of compute tasks, particularly during peak load times, e.g., at arrival of all inhabitants in the evening. Moreover, smart home applications have diverse requirements, ranging from real-time needs for *executing* tasks to non-real-time needs for *training* tasks, as well as privacy protection. These factors must be considered in offloading decisions. While existing research has focused on considerations such as latency, energy efficiency, and security (including encryption and integrity-based algorithms) [22, 23], the *privacy sensitivity of the offloaded data* has often been overlooked in this context.

As Table 1 shows, smart home applications generate and consume different data types, each holding a distinct level of privacy sensitivity.

Note that each application category includes both *training* and *executing* tasks. Therefore, applications can expose personal data about residents to a different extent, depending on the type of their data, e.g., intimate personal details about the households or their habits belonging to health and wellness category [24]. Moreover, according to European General Data Protection Regulation (GDPR), natural persons should have control of their own personal data [25, 26]. However, as Chi et al. [27] report, popular smart home platforms lack mechanisms for users to control the data sent to them. Therefore, it is critical to incorporate privacy sensitivity information into offloading decisions and empower users to participate in assessing the privacy sensitivity level of their application data.

To address this gap, we expand upon our previous work [28] and propose RASH (**R**esource **A**llocation scheme for **S**mart **H**omes). RASH operates at a home gateway (as in Fig. 1) and makes privacy-driven computation offloading decisions. While our previous study presents resource allocation and computation offloading for federated learning (FL) training tasks in smart homes, RASH generalizes and significantly extends that work, presenting a dynamic framework that jointly considers

Table 1 Smart-home application categories, highlighting the diverse data types consumed by IoT applications and the kinds of computational tasks they perform

Application category	Example application	Consumed data type	Example device
Health monitoring	Detection of cardiac disorders; detection of myocardial ischemia & infarction [9–11]; cloud–edge fall detection [12]	Personal sensing data, image	Wearable devices/sensors, camera
Energy management and optimisation	Minimising energy consumption/cost; energy-use prediction [13, 14]	Sensing data	Smart meter, thermostat
Home automation (security, safety, surveillance)	Smart-home monitoring [15]; edge-based smart-home automation [16]	Image, sensing data	Camera; gas, fire, air-quality sensors; smart lock; soil-moisture sensor; lights; fans
Threat detection and safeguarding	IoT anomaly detection and traffic filtering at the edge [17–19]	Network traffic	Any device
Entertainment and cognitive assistance	Speaker recognition at the local edge [20]; edge–cloud VR applications [21]	Audio, video	Smart speaker, TV, sound system

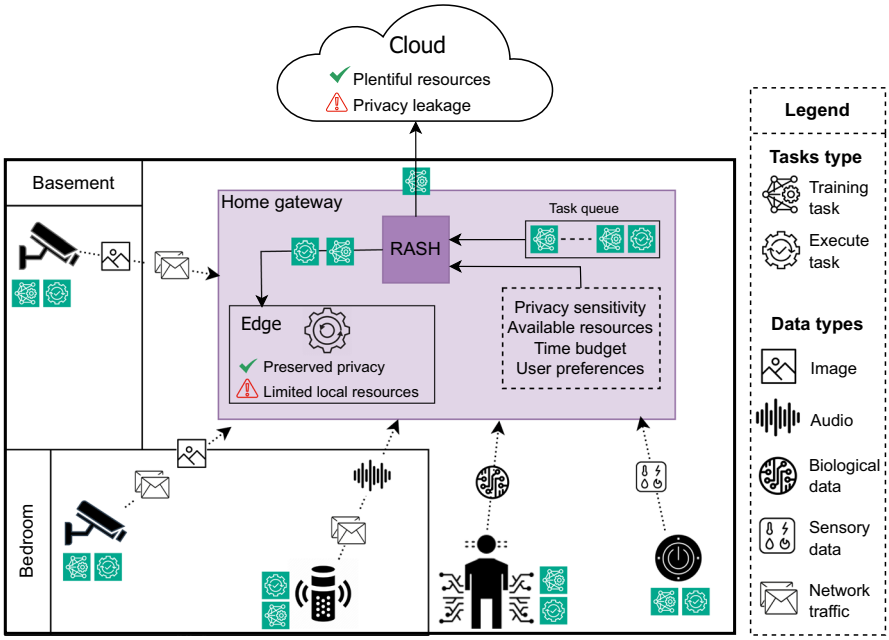


Fig. 1 A typical smart home, containing various data and task types. The logical controller, RASH, decides where to execute each task

both *training* and *executing* tasks in smart homes and supports continuous operation over time with advanced privacy score and heuristic postponing. RASH has the following three merits:

- **Privacy-driven:** RASH allocates local computational and bandwidth resources

among IoT tasks by solving an optimization problem and leverages cloud resources during high demand periods. Different from previous studies, RASH aims at minimizing the privacy risk of offloading tasks to a remote server. We quantify the privacy-sensitivity of IoT applications based on their data type and the location of IoT devices inside homes, and use this score to drive offloading decisions. Moreover, RASH facilitates user control by hosting the offloading decisions at each smart home gateway, rather than relying on the offloading policy of each smart device's cloud backend and users can adjust their privacy preferences.

- **Heuristic postponing:** When the high number of tasks and limited resources make the optimization problem infeasible, RASH uses a heuristic to postpone tasks and then solves the problem again. This process may repeat if infeasibility persists. The postponing algorithm is designed in a way that effectively postpones as few tasks as possible to maximize resource utilization and resolve the infeasibility state with fewer repeated postponements.
- **Dynamic adaptation**: RASH continuously adapts its scheduling as new tasks arrive and updates task requirements. Also, given the increasing adoption of machine learning-enabled applications, specifically FL in the smart home context, RASH considers a setting wherein IoT devices might run both training and executing tasks, which differ in terms of time budgets and computational demands. RASH models *training* tasks with parameters specific to their neural network structure. To the best of our knowledge, no prior study considers coexistence of both task types.

In this paper, we address the following research questions:

- RQ1: How can the privacy sensitivity of smart home applications be integrated into offloading decisions, and to what extent can RASH preserve user privacy compared to privacy-oblivious offloading algorithms?
- RQ2: How does computational load affect offloading decisions?
- RQ3: How do the design of the postponement algorithm and RASH affect infeasibility mitigation and the execution of *training* and *executing* tasks?

The rest of the paper is organized as follows. Section 2 reviews the most relevant work on smart homes focusing on resource management, privacy notion, and privacy-driven computation offloading while Section 3 presents the considered system model along with assumptions made for the considered setting. Next, Section 4 introduces the operation of RASH and Section 5 presents how RASH allocates resources among IoT tasks and task postponing policy which is triggered under high computation load. Next, Section 6 presents an assessment of RASH to address the raised research questions. Finally, Section 7 discusses the key limitations of our proposal and Section 8 concludes the paper listing potential future directions.

2 Related Work

This section reviews related work that propose resource management schemes specifically for smart homes, explore the concept of privacy in smart homes and measure the privacy risks associated with them. We also discuss privacy-driven offloading approaches to understand how this crucial aspect of smart homes is considered in computation offloading schemes.

2.1 Smart Home Resource Management Schemes

Resource management in smart homes includes data management, energy management, heterogeneity management, memory management, network management, etc. However, our focus here is solely on the management of computational, memory, and bandwidth resources. Various architectural solutions attempt to use the cloud resources to manage resource-constrained smart homes and propose resource allocation schemes for edge or fog nodes [29–32]. Chen et al. [33] propose an Intelligent Resource Allocation Framework (iRAF) to solve the resource allocation problem for collaborative mobile edge computing using deep reinforcement learning. iRAF solves the problem based on network status and task characteristics, such as the computing capability of edge servers and devices, communication channel quality, resource utilization, and latency requirements of the services. Samie et al. [34] propose computation offloading and bandwidth allocation for IoT devices to address under-utilization of edge bandwidth resources due to the discrete offloading levels on IoT devices. Their approach fully utilizes edge node bandwidth and improves battery life by adjusting offloading levels according to resource availability. Yang et al. [35] propose cooperative task offloading in distributed mobile edge computing systems. They introduce a multi-agent deep reinforcement learning architecture, where each edge server decides local scheduling and inter-server migration actions. Their focus is scalability of offloading as the number of edge servers grows, and they improve completion time and resource utilization. Papathanail et al. (COSMOS) [36] present an orchestration framework for task offloading for an object identification service in a smart-city deployment. COSMOS introduces workload prediction (Kalman filtering), admission control, and load balancing for improving response time and scalability of an AI inference at the edge. Sameri et al. [37] present a reinforcement learning–based orchestration approach for XR applications in distributed 6 G cloud infrastructures. Their solution uses a multi-objective reward function that jointly optimizes latency, cost, fairness, and Quality of Experience (QoE). Their work focuses on service placement for immersive XR workloads, emphasizing QoE-driven orchestration across cloud–edge resources. However, these approaches optimize for performance metrics such as latency and resource utilization, without considering privacy sensitivity in the offloading decisions.

2.2 Interpretation of Privacy in Smart Homes

Brahma et al. [38] interpret privacy in smart homes as the protection of contextual privacy, which involves safeguarding user activities from being inferred by analyz-

ing encrypted traffic metadata rather than the content itself. To address this issue, they utilized a Dynamic Traffic Shaping approach that generates dummy traffic based on the signatures exhibited by IoT devices. Bugeja et al. [39] classify smart home devices based on the type and amount of personal data they expose. Applying k-means clustering to the technical specification of 81 IoT devices listed in Mozilla's *PrivacyNotIncluded website,[1] authors classify IoT devices into four groups as *app-based accessors, watchers, location harvesters, and listeners*. PRASH [40] proposes a framework for analysing and assessing the privacy risks of smart homes. The framework involves building a smart home system model and constructing an attack tree to explore vulnerabilities of various components of the smart home model. The privacy risk score is then calculated based on the attack success likelihood and attack impact. Chhetri et al. [41] present design factors for privacy controls in smart home devices through interviews with 25 individuals. The research highlights the need for user-centric privacy controls over data collection, processing, and sharing. It suggests that smart home device designs often lack transparency and do not provide users with sufficient control or information about how their data is used and shared.

2.3 Privacy-Driven Computation Offloading

Prior works have explored issues such as location privacy and usage pattern privacy [42–44]. Several other papers focus on safeguarding privacy of the task data during offloading to different Service Providers (SPs). Razaq et al. [45] address the challenge of preserving user privacy in fog computing environments. To avoid the risk of breaches of user-privacy preservation at compromised SPs, they divide each IoT task into smaller fragments and offload them to multiple fog nodes owned by different SPs with equal or higher security credits. Peng et al. [46] present a privacy-aware offloading algorithm for Augmented Reality (AR) in healthcare systems, ensuring tasks with privacy conflicts are offloaded to different edge nodes. RT-SANE [47] presents a security-aware offloading algorithm that maximizes the success ratio of tasks while respecting tasks' security and privacy needs. They categorize tasks based on their security level and dynamically allocate them to either local micro data centers or cloud data centers based on their security needs and delay.

2.4 Novelty

While existing research highlights the importance of protecting privacy of smart home applications, most studies focus on the privacy implications of the offloading location and also lack user control over their data processing preferences. Our work addresses these gaps by integrating privacy sensitivity of applications' data into offloading and resource allocation algorithm. Moreover, our design facilitates users to have control over how their data is handled.

[1] https://foundation.mozilla.org/en/privacynotincluded/

3 System Model

We consider a smart home with N IoT devices and a home gateway, e.g., a home router, which hosts RASH and receives IoT task requests as shown in Fig. 2. The home gateway has a direct connection to a local edge device for local processing.[2] We denote the resources that RASH needs to manage as a 3-tuple: $< B, B^e, f >$ where B represents the uplink capacity of the link between the IoT devices and the home gateway, B^e is the backhaul capacity between the home gateway and the Internet, f denote the number of computing cycles of the local edge device.

In this setting, IoT devices generate *training* and *executing* tasks. In particular, we assume that all *training* tasks are FL tasks as they are favorable in smart home settings due to their privacy-by-design merits. *Executing* tasks include inference or any type of task that does not involve training. We denote IoT tasks as $\mathcal{T} = \mathcal{T}^{(t)} \cup \mathcal{T}^{(c)}$ where $\mathcal{T}^{(t)} = \{T_i^{(t)}, \cdots, T_N^{(t)}\}$ denotes the set of *training* tasks and $\mathcal{T}^{(c)} = \{T_i^{(c)}, \cdots, T_N^{(c)}\}$ denotes the set of other *executing* tasks, e.g., inference. Note that each IoT device might be associated with multiple tasks, which are differentiated by unique indexes in our model. The local edge device periodically reports the available compute capacity to RASH for its decisions. We assume that RASH is aware of the volume of data generated by IoT devices, e.g., via predicting the volume based on historical data. Table 2 summarizes the key notations used throughout the paper.

Each task is associated with a privacy-sensitivity score that indicates the sensitivity of the data being processed. Quantifying privacy-sensitivity, however, is challenging due to the lack of a standardized metric and the limited research on IoT privacy metrics. While defining such a metric is beyond the scope of this paper, we adapt the metric introduced by Bugeja et al. [39]. Since our focus is on transmission and processing of the data, we consider the sensitivity of type of data generated and the location of the IoT device, omitting the data accessibility parameter in the original model. Different types of data might have different sensitivity levels. For example, personal biological data used by healthcare applications might be more privacy-sensitive than

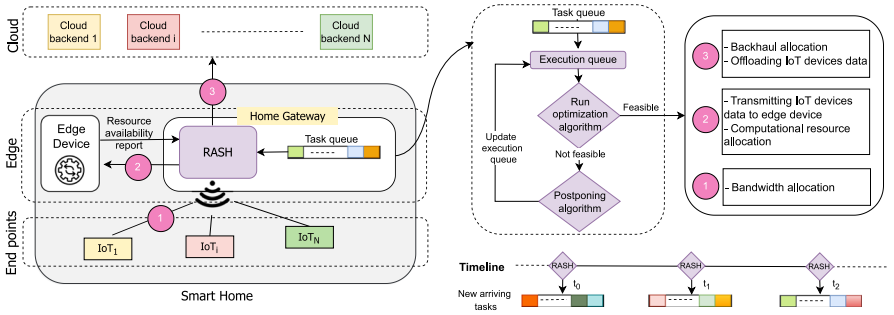


Fig. 2 Overview of RASH. The diagram illustrates the resource allocation and task offloading algorithm, which runs periodically

[2] Note that the home gateway and the edge device can also be integrated into a single unit.

Table 2 Table of notations

Notation	Description
N	Number of tasks
$\mathcal{I}$, IoT_i	Set of IoT devices and IoT device *i*
$\mathcal{T}$	Set of all IoT tasks
$\mathcal{T}^{(t)}, \mathcal{T}^{(c)}$	Set of *training* tasks, Set of *executing* tasks
$T_i^{(t)}, T_i^{(c)}$	*training* task and *executing* task of IoT_i, respectively
B, B_i	Uplink capacity between the IoT devices and the home gateway, and capacity allocated for IoT_i
B^e, B_i^e	Backhaul capacity between the home gateway and the ISP, backhaul capacity allocated to task *i*
M, M_i	Memory capacity of the edge device, the memory requirement of IoT_i
D_i	Data size of IoT_i
$D_{rem,i}$, $D_{rem,i}^{e,c}$	Remaining data size for transmission from IoT_i to the local edge, and from local edge to cloud, respectively
$t_i^{d,e}$	Required time to transmit data associated with task *i* from IoT_i to the local edge device
$t_i^{e,c}$	Required time to offload data associated with task *i* from the local edge device to the cloud
t_i^{dm}, t_i^{um}	Required time to download and upload the model associated with $T_i^{(t)}$, respectively
t_i^e	Required time to executing task *i* on the edge device
t_i^l	Required time for completing task *i* locally
t_i^r	Required time for completing task *i* remotely
$t_i^{\max}$	Delay budget of task *i*
t_i^{rem}	Remaining time budget of task *i*
f	# of CPU cycles per second of the local edge device
f_i	# of CPU cycles allocated to task *i*
C_i^e	# of CPU cycles required for executing task *i* on the local edge device
N_i^c	# of CPU cycles required to train one bit data associated with $T_i^{(t)}$ on the local edge device
η_i, S_i	# of epochs for training $T_i^{(t)}$ and model size of $T_i^{(t)}$, respectively
p_i	The privacy-sensitivity level of task *i*

sensory data collected by a temperature or fire sensor. Based on the application categories in Table 1, we identify five data types used by IoT devices for their computation: (1) image, (2) audio, (3) sensing data, (4) biological data, (5) network traffic. Moreover, the location where the IoT device resides also reflects the privacy level of the relevant task to that IoT device. As an example, the bedroom is likely to have a higher privacy level compared to the basement.

Let us denote by q_j the privacy parameter where $j = \{1, 2\}$ represents the data type sensitivity and location sensitivity, respectively. Moreover, let us denote by $\omega(T_i, q_j)$ the sensitivity level of T_i with respect to the parameter q_j. Then, we define the privacy sensitivity score of task *i* as follows [39]: $p_i = \sum_{j=1}^{k} \omega(T_i, q_j)$ where k denotes the number of privacy parameters, which in our case is two (data type sensitivity and location sensitivity). In the case when an application has multiple types of data, the privacy score of that application can be calculated as the maximum privacy

score among all data types involved.[3] However, for simplicity we assume that each application is associated with only one type of data. Note that privacy parameters can be tuned based on context, e.g., users can dynamically adjust the value for each privacy parameter based on their privacy perceptions and preferences. The resulting privacy score p_i can only take discrete values.

Consequently, let us define a *training* task as follows:

$$T_i^{(t)} =< D_i, S_i, \eta_i, N_i^c, t_i^{\max}, m_i, p_i > \tag{1}$$

where:

- D_i denotes the size of the data generated by IoT_i related to task $T_i^{(t)}$, (Mb).
- S_i denotes the size of the model parameters related to task $T_i^{(t)}$, (Mb).
- η_i denotes the number of epochs required for training the learning model of task $T_i^{(t)}$.
- N_i^c denotes the number of CPU cycles required to train one bit data of task $T_i^{(t)}$.
- $t_i^{\max}$ denotes the delay budget for each round of training associated with $T_i^{(t)}$. Although *training* tasks are typically time insensitive, when it comes to FL, each *training* task must be completed below a certain time budget to avoid the problem of *stragglers* [48]. Stragglers are participants in an FL process that do not deliver their data on time, e.g., due to insufficient computation or communication capacity. Consequently, training may take longer and affect the FL accuracy and convergence time. We assume that this delay budget is determined by the FL model aggregator to prevent the stragglers effect.
- m_i denotes the required memory for executing the task and storing intermediate parameters until finishing the task. We assume that computation requires moving all data associated with a task to the memory. Hence, the total memory for processing each task is modeled as the sum of its data size and the required memory capacity, i.e., $M_i = D_i + m_i$.
- p_i denotes the privacy-sensitivity score of $T_i^{(t)}$.

We denote *executing* tasks by a 4-tuple as follows:

$$T_i^{(c)} =< D_i, C_i^e, t_i^{\max}, m_i, p_i > \tag{2}$$

where C_i^e denotes the number of CPU cycles required for executing task $T_i^{(c)}$ on the local edge device. The rest of the parameters are the same as defined for *training* tasks.

[3] Other approaches, such as summing the scores, are also viable, each with its own advantages and drawbacks. We leave exploration of these alternatives for future work.

4 Overview of RASH

We assume a time-slotted system with a slot duration of Δt where at the beginning of each slot RASH determines the bandwidth and compute resources for each IoT task by solving an optimization problem (introduced in Section 5). Let us denote by B_i the uplink capacity allocated for IoT_i, by B_i^e the allocated backhaul capacity, and by f_i the number of CPU cycles assigned.

When RASH runs for the first time, it moves all tasks from the queue to the execution queue and allocates resources and determines the offloading strategy for each task. Tasks arriving to the home gateway are stored in a queue till the next time slot. At the start of the next time slot, RASH updates remaining time budget and computational requirements for uncompleted tasks. Then RASH puts both newly arrived and incomplete tasks into the execution queue, to allocate resources and decide on their offloading strategy. If RASH fails to find a feasible solution, it postpones some tasks based on their remaining time budget (according to the task postponing policy introduced in Section 5), moves them back to the task queue and runs again with the updated execution queue.

Computation can be performed *locally* (i.e., at the local edge device) or *remotely* (i.e., at a cloud or fog device) depending on the available resources and task requirements. Since tasks have latency budget and the location of execution affects the task completion time, next we present the task completion time based on B_i, f_i, and B_i^e considering *local processing* and *remote processing*.

- *Local processing*: First, we explain the required time for local processing of the *training* tasks which consists of four steps: time for data transmission from IoT device to the local edge device ($t_i^{d,e}$), time for local processing (t_i^l), time for uploading the trained model (t_i^{um}) and time for downloading the global model (t_i^{dm}). The time required to transmit the generated data D_i from IoT_i to the local edge device is simply calculated as follows:

$$t_i^{d,e} = \frac{D_i}{B_i} \quad [\text{seconds}]. \tag{3}$$

The required number of CPU cycles for processing the *training* tasks on the local edge device (C_i^e) depends on several parameters: number of epochs η_i for training $T_i^{(t)}$ model, size of the data to be processed locally (D_i^e), and the number of CPU cycles required for training one bit data of $T_i^{(t)}$ (N_i^c). We then calculate C_i^e as follows [49]:

$$C_i^e = \eta_i D_i^e N_i^c \ [\text{cycles}]. \tag{4}$$

Therefore, the time required to complete local training of $T_i^{(t)}$ can be calculated as follows:

$$t_i^e = \frac{C_i^e}{f_i} \quad [\text{seconds}]. \tag{5}$$

Since FL training is a collaborative training approach, each trained model should be sent to the aggregator server when the training is completed. In fact, the time budget set for completing FL training tasks also includes the time for sending the trained model to the server. Therefore, we calculate the required time for sending the model parameters with size S_i to the server as follows:

$$t_i^{um} = \frac{S_i}{B_i^e} \quad [\text{seconds}]. \tag{6}$$

The required time for downloading the neural network model associated with $T_i^{(t)}$ is considered as a constant value since it depends on the downlink bandwidth between the local edge device and the server. Finally, the required time for performing one round of local training of $T_i^{(t)}$ is as follows:

$$t_i^l = t_i^{d,e} + t_i^e + t_i^{um} + t_i^{dm} \quad [\text{seconds}]. \tag{7}$$

As for the *executing* tasks, $t_i^{d,e}$ can be calculated similarly. Then, the time required to complete the execution of the task is as follows:

$$t_i^e = \frac{C_i^e}{N_i^c} \quad [\text{seconds}]. \tag{8}$$

Unlike *training* tasks, *executing* tasks do not need to send or receive any data from remote servers for their local execution. Thereby, the required time for completing $T_i^{(c)}$ locally is as follows:

$$t_i^l = t_i^{d,e} + t_i^e \quad [\text{seconds}]. \tag{9}$$

- *Remote processing:* In this case, once the data is transmitted to the local edge device, it must be further uploaded to the remote server. Hence, while $t_i^{d,e}$ remains the same, now we need to account for the delay of sending the data to the remote server. This delay is calculated for both *training* and *executing* tasks in the same way as follows:

$$t_i^{e,c} = \frac{D_i}{B_i^e} \quad [\text{seconds}]. \tag{10}$$

Since the cloud is an compute-rich environment, we assume that the required time to process data on the cloud is negligible. Finally, the total delay for completing the task remotely is calculated as follows:

$$t_i^r = t_i^{d,e} + t_i^{e,c} \quad [\text{seconds}]. \tag{11}$$

5 Resource Allocation for Smart Homes Considering IoT Task Type and Privacy Sensitivity

RASH aims at minimizing the privacy risk of offloading IoT tasks for privacy preservation while meeting the delay budget of each task. For its operation, we define the decision variables as B_i, B_i^e, f_i, and $\alpha_i \in \{0,1\}$ which denotes the local or remote execution of the task. We denote the privacy risk of offloading an IoT task by r_i which depends on the value of α_i and the privacy-sensitivity score of that task. Therefore, we will have:

$$r_i = \begin{cases} p_i & \alpha_i = 1, \ [\text{Remote processing}] \\ 0 & \alpha_i = 0, \ [\text{Local processing}] \end{cases} \tag{12}$$

where α_i can be set by users per IoT application, allowing them to set their preferences for local or remote execution of applications. It is worth noting that RASH keeps the initially decided value of α_i for each task when they encounter the RASH algorithm for the first time and does not reassign the value of α_i for those tasks in the next time slots. Therefore, the offloading strategy for each task always remains the same across multiple executions of RASH over time.

Now, let us formulate the resource allocation for IoT tasks:

$$\mathcal{P}_1 : \min_{\alpha_i, B_i, f_i, B_i^e} \max \ r_i \tag{13}$$

$$\text{subject to: } \alpha_i t_i^r + (1-\alpha_i) t_i^l \leqslant t_i^{\max} \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{14}$$

$$\sum_{i=1}^{N} f_i \leqslant f \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{15}$$

$$\sum_{i=1}^{N} B_i^e \leqslant B^e \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{16}$$

$$\sum_{i=1}^{N} B_i \leqslant B \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{17}$$

$$\sum_{i=1}^{N} M_i (1-\alpha_i) \leqslant M \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{18}$$

$$\alpha_i \in \{0,1\} \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{19}$$

$$f_i, B_i, B_i^e \in \mathbb{R} \quad \forall \mathcal{T}_i \in \mathcal{T} \tag{20}$$

where (13) minimizes the maximum value of risk score that each IoT task can hold for mitigating the user privacy loss. Constraint (14) ensures that the delay restriction defined for each IoT application is satisfied. Constraint (15), (16), and (17) guarantee that all resource limitations (i.e., uplink, backhaul, and CPU cycles, respectively) are considered in resource allocation. Constraint (18) ensures that tasks selected for local processing (i.e., with $\alpha_i = 0$) can fit into the memory. Constraint (19) defines the domain of α_i. Finally, (20) denotes that the remaining decision variables are continuous. To solve this mixed integer problem, we use an optimization solver and find the optimal values of $\alpha_i, B_i, f_i, B_i^e$. Once RASH solves the optimization problem, it updates remaining time budget, compute needs and remaining data to transfer for all tasks based on the decision variables. The steps of RASH in one iteration are summarized in Algorithm 1.

Input: $Q = \{\mathcal{T}_i : (t_i^{rem}, \alpha_i, C_i^e, D_{rem,i}, D_{rem,i}^e),\ i = 1, 2, \ldots, N\}$ ▷ Set of tasks with their specifications
Output: Q ▷ Set of tasks with updated values based on decision variables

$exec\_Q \leftarrow Q$
$res \leftarrow Solver(exec\_Q)$ ▷ Solves $\mathcal{P}_1$ (13)
while $res.status$ **is** infeasible **do**
    $exec\_Q \leftarrow$ POSTPONING$(exec_Q)$
    $res \leftarrow Solver(exec\_Q)$
end while
for $\mathcal{T}_i$ **in** Q **do**
    t_i^{rem} $-=$ Δt
end for
for $\mathcal{T}_i$ **in** $exec\_Q$ **do**
    if α_i **is not set then**
        $\alpha_i \leftarrow res[i].\alpha_i$
    end if
    C_i^e $-=$ Δt . $res[i].f_i$
    $D_{rem,i}$ $-=$ Δt . $res[i].B_i^e$
    $D_{rem,i}^{e,c}$ $-=$ Δt . $res[i].B_i$
end for
return Q

Algorithm 1 RASH

5.1 Task Postponing Policy

In case of high demand and under strict performance requirements, RASH might initially fail to find a feasible solution and needs to postpone one or more tasks to the consequent time slot(s). After postponing, RASH runs again to find a feasible solution. If it remains infeasible, this process repeats until finding a feasible solution.

1: **Input:** Task queue (Q)
2: **Output:** Execution queue ($exec\_Q$)
3:
4: sorted_Q $\leftarrow$ Sort tasks in Q based on t_i^{rem}
5: exec_Q $\leftarrow$ []
6: $f_{exec} \leftarrow 0$, $B_{exec} \leftarrow 0$, $i \leftarrow 0$
7: **while** $f_{exec} < f$ **and** $i <$ length(sorted_Q) **do**
8: $task_i \leftarrow$ sorted_Q[i]
9: **if** $\alpha_i \,!= 1$ **then**
10: exec_Q.append($task_i$)
11: $f_{exec} \mathrel{+}= \frac{c_i^e}{t_i^{rem}}$
12: sorted_Q.pop(i)
13: **else**
14: $i \leftarrow i + 1$
15: **end if**
16: **end while**
17: **while** $B_{exec} < B^e$ **and** $i <$ length(sorted_Q) **do**
18: $task_i \leftarrow$ sorted_Q[i]
19: **if** $\alpha_i == 1$ or $\alpha_i == -1$ **then**
20: exec_Q.append($task_i$)
21: $B_{exec} \mathrel{+}= \frac{D_i}{t_i^{rem}}$
22: sorted_Q.pop(i)
23: **else**
24: $i \leftarrow i + 1$
25: **end if**
26: **end while**
27: **return** exec_Q

Algorithm 2 POSTPONING (Q)

The task postponing process, outlined in Algorithm 2, works as follows: first, all tasks are sorted by their remaining time budget (line 4). Then tasks with lowest remaining time are selected for execution (except those decided for remote execution), until their combined compute needs per second adds up to f (lines 7–16):

$$j^* = \arg\max_j \left\{ \sum_{i=1}^{j} \frac{C_i^e}{t_i^{rem}} \leq f \right\} \quad 1 \leq j \leq N. \tag{21}$$

We consider the remaining time budget as an approximation for the time that could be allocated for processing. However, this is an optimistic approximation because the remaining time budget also includes the time needed for data transmission (albeit minimal due to small data sizes and ample bandwidth), resulting in shorter CPU time and potentially requiring more computation, which could exceed f. However, this allows us to ensure maximum utilization of computing resources. On the other hand, offloading to remote servers might not happen as all the selected tasks can be optimistically served locally. To leverage the potential benefits of remote servers and ensure backhaul utilization, we start selecting tasks among the remaining tasks that were already decided for remote execution or are not yet decided upon, and add them

to the execution queue (for newly arrived tasks, α_i = -1, representing not set yet). The selection is carried out in such a way that the total required backhaul bandwidth per second for the chosen tasks, represented by $\sum_{i=1}^{j} \frac{D_i}{t_i^{rem}}$, is smaller or equal to B^e (lines 17–26). Tasks with the tightest time budget are prioritized for this selection.

6 Performance Analysis

In this section, we evaluate the performance of our proposal under various settings and compare with two baselines that do not consider privacy leakage in their objective function to address the research questions listed in Section 1. In the following, we first introduce our assumptions and parameters. Afterward, we present and discuss the results of our extensive simulations.

6.1 Simulation Setting

Let us first present considered parameters for our evaluation, studied scenarios and performance metrics.

Home gateway: We consider a home Wi-Fi router with a backhaul capacity of 40 Mbps[4] and 150 Mbps uplink/downlink capacity as in Wi-Fi 802.11n. The local edge device is equipped with a 1 GHz compute capacity and 4 GB of RAM, similar to common edge devices in smart homes such as Raspberry Pis.

6.1.1 IoT Tasks

For *executing* tasks, we set $D_i \sim [1, 1000]$ kB, covering applications with different types and sizes of data, including video, audio, and sensory data. We initialize z_i within [50, 500] Megacycles, covering wide range of applications in terms of computational complexity. *Executing* tasks encompass a wide variety of tasks with various end-to-end latency, from tens of milliseconds to seconds [21, 50, 51]. Thus, we assume time budgets between [0.01, 5] seconds. We set the memory usage of execute tasks within [10, 100] MB [52]. For *training* tasks, we consider the following parameters. A typical FL training round may last anywhere from 1 to 10 min [53]. However, training round time could be variable depending on the FL task, e.g., 2–3 min for a next word prediction task [54]. In our study, since we run the simulation for 20 iterations for each analysis, we limited the duration of simulation to 1 min. Therefore, we set the time budget of *training* tasks to 1 min and assume they all arrive at time slot zero. The size of neural network models varies depending on the application, ranging from a simple image classification model with a total of 199,210 parameters to a next-word prediction model with 4,950,544 parameters [55]. To accommodate this variability, we assume that $S_i \sim [3, 120]$ Mbits. Considering that training operations involve processing stored data, which typically has larger sizes compared to *executing* tasks, we assume training data sizes ranging from 0.1 to 100 MB. The required

[4] https://www.speedtest.net/global-index.

CPU cycles for training 1-bit data highly depends on the model size. Therefore, we use a linear function to calculate the required CPU cycles for training 1-bit data, with a coefficient determined based on the required CPU cycles for a 40Mb model size, which was reported as 500 in [56]. Although training tasks are memory-intensive, techniques such as model quantization and optimized operator execution order [57–60] have been widely used to lower the memory requirements of these tasks and run them on resource-constrained devices with significantly lower memory consumption. We consider the memory consumption of training tasks between [100, 500] MB [61]. The number of epochs is uniformly distributed between 1 and 10.

As for the privacy-sensitivity score, we define $\omega(T_i, q_j)$ for data type sensitivity parameter as follows: personal biological data$= 5$, image$= 4$, audio$= 3$, network traffic$= 2$, sensing data$= 1$. For the location parameter, we define the following zones within a smart home: living room, bedroom, kitchen, and basement. Afterwards, we assign the following weights for each zone: bedroom$= 4$, living room=3, kitchen$= 2$, basement$= 1$. According to our predefined weights, privacy-sensitivity scores fall within the range of [2, 9] and we randomly initialize task privacy scores with discrete values from this range. Other initialization values are also possible, especially where certain data types hold significantly greater sensitivity level relative to others. However, exploring these alternatives and determining which values to assign is beyond the scope of this research.

Our simulator is publicly available for further development[5]. Using our Python simulator with Gurobi[6] solver, we run each studied scenario for one minute over 20 iterations while initializing $\Delta t = 0.01$ second.

6.1.2 IoT Traffic Model

We aim to replicate real-world scenarios in smart homes, particularly load spikes when residents return home in the evening. Therefore, we run our algorithm under various system loads ($n\%$), over 60 seconds. A system load of $n\%$ indicates that the cumulative computational demand of all current tasks in the system equals $n\%$ of the available local computational resources. At the simulation start, we randomly generate tasks with computational requirements totaling $n\%$ of the total computational resources. The system load is dynamically updated based on unfinished tasks. At the beginning of the next time slots if the system load drops below $n\%$, new tasks are added to maintain the desired level. We assume that tasks enter the system only at the beginning of the time slots. We analyze system loads from 10% to 130%, representing low to peak loads. Note that, due to the dynamic nature of the simulation, fixing the system load at a specific level is impractical. Instead, the system load dynamically adjusts to accommodate around the intended load level. The number of tasks generated varies with iteration and system load, ranging from 70-200 tasks at low loads until 70% , 200-400 at loads until 100% load and 400-600 at loads above 110%, for the whole simulation period. Note that all generated tasks are feasible for both local computation and remote execution, meaning that their computational demands rela-

[5] https://github.com/Tina-Rezaei/RASH

[6] www.pyomo.org and https://www.gurobi.com

tive to their time budgets do not exceed the available local resources, and their data sizes, likewise relative to their time budgets, remain within the limits of the available backhaul capacity.

6.1.3 Baseline

To evaluate different aspects of RASH and investigate the effect of our objective function in (13), we introduce the following methods as benchmark:

- ***MinMaxDelay***: A variant of RASH that does not consider privacy. The objective of *MinMaxDelay* is to minimize the response time of tasks using the following objective function: $minmax \quad \alpha_i t_i^r + (1-\alpha_i) t_i^l$, and is subjected to the same constraints in (14)-(20). We selected this variant since it is prevalent in the literature [42, 45, 46].
- **QEOS** [62]: A distributed deep reinforcement learning-based task offloading algorithm designed for mobile edge computing (MEC) environments. QECO maximizes user-specific Quality of Experience (QoE) by balancing task delay, completion rate, and energy consumption. However, QECO does not consider data privacy.

6.1.4 Performance Metrics

In our performance assessment, we use the following metrics:

- *Fraction of satisfied tasks* measures the proportion of tasks among N tasks that are completed within their time budget. If a task's time budget is exhausted, but the task is not yet complete, we mark it as an *unsatisfied task.*
- *Time budget utilization* measures the proportion of time budget that is consumed by the task when it is completed.
- *CPU utilization*. Measures the proportion of local edge device CPU cycles used during a 1-minute execution of RASH.
- *Backhaul bandwidth utilization* measures the proportion of backhaul used during a 1-minute execution of RASH.
- *Energy consumption* measures the total energy consumed by the system. To enable a fair comparison with QECO [62], we adopt the same energy consumption model introduced in their work which includes edge device computation, data transmission, remote server processing, and device standby energy consumption.

6.2 Analysis of the Results

6.2.1 Privacy Preservation Analysis

To answer RQ1, we investigate the effect of our algorithm on the privacy preservation of IoT applications. We compare in Fig. 3 the performance of RASH with *MinMaxDelay* to examine how different objectives affect privacy preservation during offloading under various load settings. The figure demonstrates the percentage

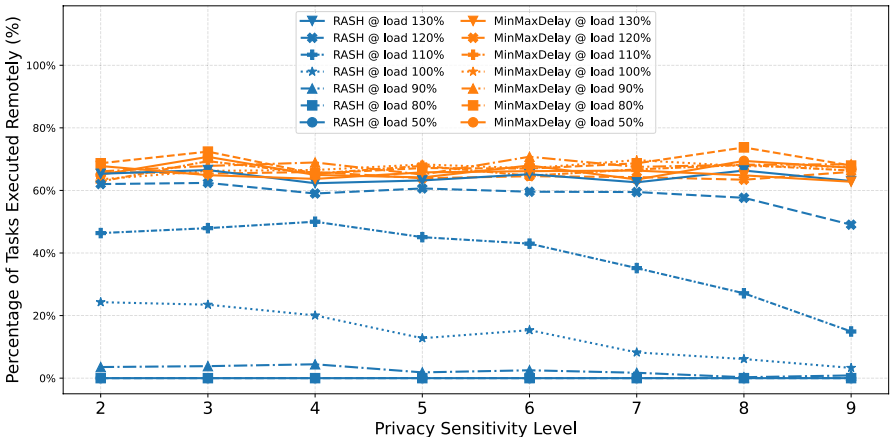


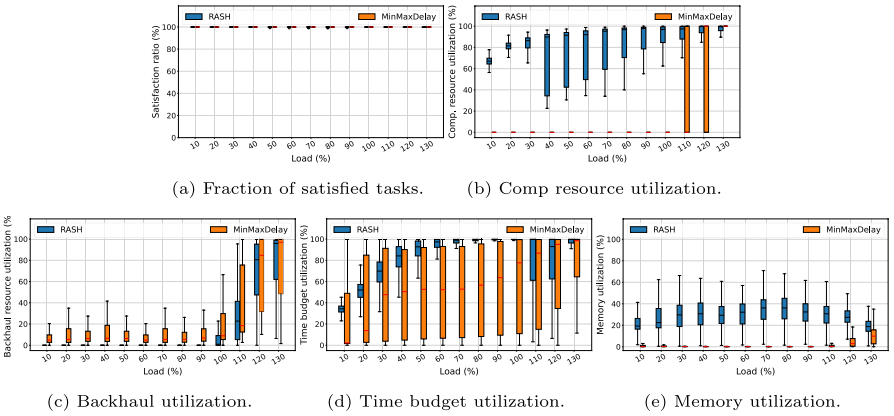
Fig. 3 Percentage of remotely executed tasks separated by privacy-sensitivity levels over 20 iterations for RASH and its variant *MinMaxDelay*

of offloaded tasks in each privacy-sensitivity level. When the system load is at or below 90%, RASH executes all tasks locally since the local resources are sufficient. In contrast, *MinMaxDelay* offloads as many tasks as possible to optimize its objective function, even when all tasks could be processed locally. As the load increases and local computational resources become saturated, RASH begins to offload tasks, predominantly those with lower privacy sensitivity. For example, at load 110%, only 15% of the tasks with the highest sensitivity level are offloaded, compared to around 50% of tasks within each sensitivity level lower than 4. This selective offloading highlights how RASH prioritizes tasks based on their privacy requirements when system is saturated and not able to process tasks locally. When the system is highly overloaded, at loads 120% and 130%, we do not observe a pronounced prioritization in offloading. This is because when new tasks arrive, the local computational resources are already fully occupied by the current tasks (recall that once α_i is set, it is not updated later), thereby forcing the offloading of newly arrived tasks regardless of their privacy sensitivity level. However, we still observe a slightly lower percentage of higher privacy-sensitive tasks being offloaded compared to other levels. On the other hand, *MinMaxDelay* consistently offloads a large fraction of tasks from all sensitivity levels and since it is privacy-oblivious, we do not observe any difference in offloaded fraction across different privacy sensitivity levels.

6.2.2 Local Resource Utilization

To answer RQ2, we investigate the effect of computational load on satisfaction ratio, CPU utilization, backhaul utilization, time budget utilization and memory consumption.

Figure 4a demonstrates the satisfaction ratio under various loads. RASH and *MinMaxDelay* consistently satisfy almost 100% of the tasks. This is because both the local computational resources and the backhaul capacity are sufficient to meet the demands of all tasks, allowing them to be either executed locally or offloaded.



(a) Fraction of satisfied tasks. (b) Comp resource utilization.

(c) Backhaul utilization. (d) Time budget utilization. (e) Memory utilization.

Fig. 4 Impact of increasing computation load

Fig. 4b compares CPU utilization between RASH and *MinMaxDelay* under various computation loads. RASH always prioritizes local execution and in case of insufficient resources utilizes backhaul to offload tasks. Therefore, under low loads, as expected, RASH does not fully utilize the CPU. For instance, at a 10% load, it utilizes approximately 67% of the CPU. Although the system load is low, RASH allocates more resources to complete tasks earlier than their time budget. As the load increases, CPU utilization also rises, reaching nearly 100%, indicating that RASH processes as many tasks locally as possible.

Meanwhile, *MinMaxDelay* favors offloading by default and chooses local computation when backhaul resources are limited. Therefore, *MinMaxDelay* results in minimal local computational utilization under low loads. For instance, when the load is lower than 100%, CPU utilization is nearly zero. Note that, *MinMaxDelay* consumes the same set of tasks processed by RASH, its preference for remote execution leads to lower number of tasks processed locally. Therefore, the CPU is utilized only for a few time slots, and for the rest remains free, making its utilization nearly invisible in the box plot. However, as the system load increases, the backhaul becomes saturated, preventing further offloading. In these cases, *MinMaxDelay* rely more on local resources, leading to a noticeable rise in CPU utilization.

We further investigate the backhaul utilization under various loads in Fig. 4c . At loads below 100%, RASH processes all tasks locally, resulting in almost zero backhaul utilization. On the other hand, *MinMaxDelay* consistently uses backhaul since it always offloads most of the tasks. However, its utilization remains low under lighter loads (below 100%) due to the low number of generated tasks and abundant backhaul capacity. As the load surpasses 100%, backhaul usage increases for both methods, driven by a higher number of generated and offloaded tasks.

Figure 4d shows time budget utilization of all tasks. As Fig. 4d illustrates, *MinMaxDelay* mostly finishes tasks quicker than RASH. This is because *MinMaxDelay* prioritizes offloading as it is faster, leading to faster task completion. At low loads, both approaches show lower time utilization due to the abundant available resources, allowing tasks to receive more resources and finish faster. As the load increases and resource contention intensifies, task completion times also rise for both methods.

However, at load 110%, RASH shows a decrease, attributed to the offloading of a noticeable number of tasks, which results in a faster completion time for offloaded tasks. However, by increasing the load further, time budget utilization begins to rise again, as both computing and backhaul resources become nearly fully utilized. Despite RASH taking longer, it completes the tasks within their time budget and satisfies all tasks. However, the rapid completion by *MinMaxDelay* comes at the cost of sacrificing privacy since all tasks are offloaded.

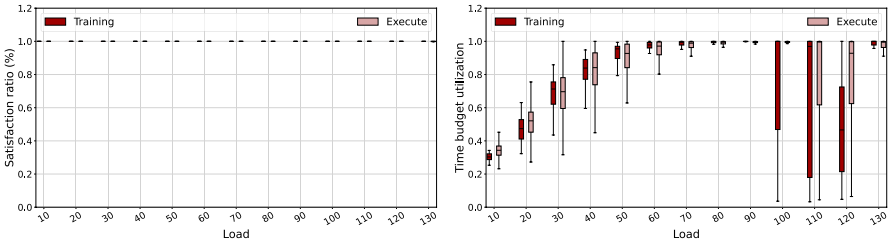
Figure 4e shows the memory consumption of the edge device. RASH consistently consumes more memory than *MinMaxDelay* due to its higher reliance on local processing. In contrast, *MinMaxDelay* maintains minimal memory usage, which only increases slightly at the highest loads (120% and 130%) when it has to process more tasks locally. Overall, we see maximum of around 37% utilization of memory since IoT tasks typically have low memory demands. Even training tasks do not impose heavy memory requirements, thanks to techniques such as model quantization and weight pruning. RASH 's memory usage drops at higher loads. This is because the system experiences more frequent postponements, especially for training tasks. As a result, many of these tasks can no longer meet their deadlines locally and are instead offloaded. Given that training tasks require significantly more memory than execution tasks, we see a noticeable decrease in memory usage at high loads.

6.2.3 Postponement Impact

To answer RQ3 first we discuss the performance of postponement algorithm under varying computational loads and then compare the completion of training and executing tasks.

When the system load was lower than 100%, no postponement was triggered. At load levels exceeding 100%, 91% of infeasibility events were resolved with at most a single postponement round. Postponing algorithm was called 1% and 3% of the decision-making points on average, for loads 110% and 120%, respectively. However, as the load increases to 130%, this frequency rises to 70%, indicating that the postponing algorithm is triggered most of the times with an average of 25% of tasks being postponed per decision point. This is expected since system is overloaded and cannot handle all the tasks, necessitating the postponement of some tasks. Generally, in almost all decision-making points where postponement is required, we observe that all the postponed tasks are *training* tasks. This is expected since *training* tasks have considerably longer time budget than *executing* tasks. In some cases where number of postponed tasks is higher than number of *training* tasks then some of the *executing* tasks are also postponed. Overall, our heuristic postponing algorithm can effectively identify tasks that should be postponed.

we compare the satisfaction ratio and time budget utilization of *training* and *executing* tasks to understand how RASH affects the execution of different task types. Note that *training* tasks are only generated in the first 10 s of the simulation due to their long time budget and the limited simulation runtime. Therefore, the number of *training* tasks is considerably lower than *executing* tasks, mostly 3 to 6 *training* tasks in each iteration. Fig. 5a compares the satisfaction ratio. RASH satisfies all training and executing tasks. This indicates that, despite limited resources and rising conten-



(a) Fraction of satisfied tasks. (b) Time budget utilization.

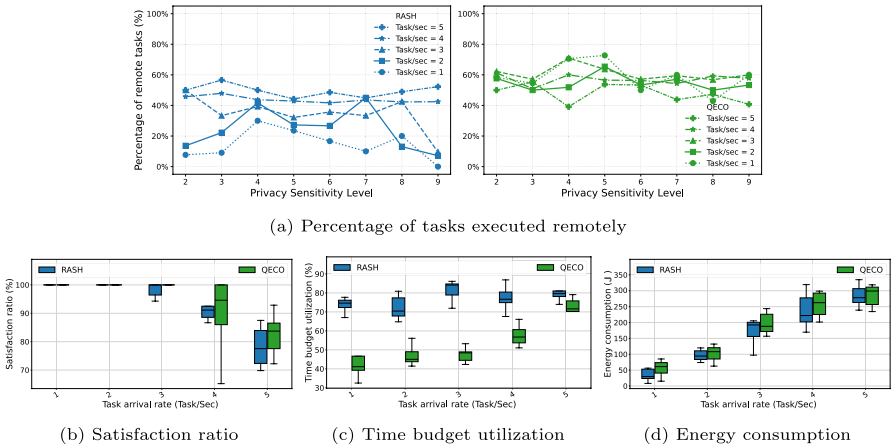
Fig. 5 Comparing training and executing tasks under increasing computation load

tion, RASH ensures task completion for both types of tasks. Fig. 5b compares time budget utilization between *training* and *executing* tasks. As expected, utilization rises with load for both training and executing tasks. Similar to the general trend observed earlier (Fig. 4d), we see a drop when the system begins to become overloaded. This effect is more pronounced for training tasks, whose extended time budgets make them more susceptible to postponement and eventually offloading. As system load continues to rise and both computational and backhaul resources become fully utilized, time budget utilization increases again for both task types. while all tasks are ultimately completed, executing tasks benefit from earlier resource allocation due to their lower time budget, while training tasks tend to be deferred. Although this behavior is aligned with satisfying all deadlines, it suggests an opportunity for enhancing RASH with fairness-aware mechanisms to ensure more balanced share of resources among different types of the tasks under heavy overload conditions.

6.2.4 Comparison to QECO [62]

In this part we compare the performance of RASH against the reinforcement-learning baseline QECO. Since QECO requires thousands of training episodes to reach convergence, we report its results only after convergence has been achieved. To ensure a fair evaluation, we adopt the exact configuration parameters reported in the original QECO study, with the exception of reducing the number of user devices to one, matching our setup, which models a single smart home controller. We execute QECO for 10 independent iterations for each task arrival ratio and evaluate RASH over the same 10 iterations. We compare the two approaches regarding their privacy preservation and across the metrics used in the QECO study, including time budget utilization, energy consumption, and satisfaction ratio. We perform a comparison across various task arrival rates, ranging from 1 to 5 tasks per second. It is important to note that, unlike our setup, in their configuration even an arrival rate of 5 tasks per second imposes a high load on the system due to the large size of the tasks.

RASH focuses on privacy-aware offloading, aiming to minimize task exposure while balancing performance. QECO, in contrast, jointly optimizes satisfaction ratio, time budget utilization, and energy consumption. Figure 6a compares the performance of the two approaches in terms of privacy preservation. RASH consistently offloads a smaller fraction of tasks across all loads. While RASH keeps the offloading ratio below 50% for most of the task arrival rates across all sensitivity levels, QECO



(a) Percentage of tasks executed remotely

(b) Satisfaction ratio (c) Time budget utilization (d) Energy consumption

Fig. 6 Performance comparison between RASH and QECO

frequently exceeds that threshold. For instance, even under a light load (arrival rate of 1), QECO offloads over 50% of tasks at each sensitivity level, whereas RASH keeps the ratio below 30%. Although RASH does not show a strong downward trend in offloading as sensitivity increases, there is a slight trend showing reduced offloading for highly sensitive tasks (e.g., level 9), particularly under lighter loads.

Figure 6b demonstrates satisfaction ratio under various task arrival rates. Under lighter loads (task arrival rate below 4), RASH and QECO achieve similar task satisfaction ratios. With increasing load, the satisfaction drops for both as the system becomes overloaded. However, QECO shows better performance, satisfying 4% to 7% more tasks on average, under the task arrival rate of 4 and 5, respectively. Fig. 6c further shows that QECO completes tasks earlier than RASH as it optimizes the task delay in its objective, resulting in the offloading of more tasks. However, this earlier completion comes at the cost of privacy exposure. When the load increases, this advantage diminishes since both methods begin to fully utilize computational and backhaul resources. Fig. 6d compares the energy consumption. The overall energy consumption increases for both approaches as the number of tasks increases. We observe that in most of the task arrival rates RASH consumes less energy, up to $50J$ less compared to QECO. This suggests that under the given configuration, the energy required for task transmission and edge execution outweighs the local processing cost. In summary, while QECO achieves higher satisfaction under heavy loads and lower delays, RASH offers a more privacy-conscious alternative, processing more tasks locally and keeping highly privacy-sensitive tasks local.

7 Discussion and Limitations

In this section we discuss the computational complexity and limitations of our algorithm.

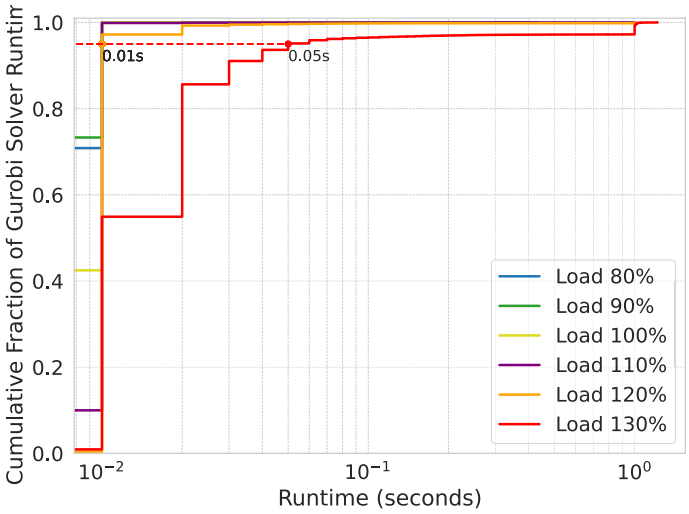


Fig. 7 Cumulative distribution function of the solver runtime throughout all decision making points over various load settings

7.1 Computational Complexity

The computational complexity of RASH is determined by the number of postponement attempts and the complexity of optimization and postponement algorithms. The optimization problem formulated in RASH is a Mixed-Integer Quadratic Program (MIQP), which is known to be NP-hard [63] and the theoretical worst-case complexity is exponential in the number of binary decision variables. While postponing policy can run in $O(|\mathcal{T}|\log|\mathcal{T}|)$ complexity, the computation is dominated by solving $\mathcal{P}_1$ in (13).

In our analysis, we use the Gurobi solver, which reduces the complexity significantly through techniques such as presolve, cutting planes and heuristics [64]. Therefore, we empirically evaluate the practical complexity of our decision-making algorithm during runtime. We record the solver runtime across all experiments and load settings, which is available in Gurobi log. A large fraction of instances were solved at the root node (zero branches), resulting in run times of effectively zero seconds. Fig. 7 shows the cumulative-distribution curve of RASH 's optimization runtime across all time slots and load levels. Note that load levels below 80%, the solver's runtime was almost zero and therefore not visible in the plot. The dashed horizontal line at the 95^{th} percentile highlights that for load levels under 130%, 95% of the optimization runs complete in under 10 ms. At a 130% load, this threshold increases to approximately 50 ms, yet over 50% of the runs still finish within 10 ms. At load 130%, although this number increases to 50 ms, still more than 50% of the runs complete within 10 ms. There are also some rare cases that the runs take up to 1 s. However, these cases can be effectively bounded by imposing a maximum time limit on the solver to ensure predictable performance. This observation acknowledges the practicality of RASH's decision-making algorithm even under high load settings. For the postponement algorithm, as discussed, postponing algorithm is called only under

high loads, with 91% of infeasibility events resolved by at most a single postponement round. Therefore, the complexity added by the postponing algorithm remains negligible. Note that Δt can also be adjusted.

7.2 Limitations

Various parameter initializations for privacy-sensitivity might affect offloading decisions. In our analysis, the privacy sensitivity of applications increases linearly relative to each other. However, other methods are possible, for example where certain data types such as personal biological hold significantly greater sensitivity relative to others. In this case we expect to see fewer high-privacy-sensitive tasks being offloaded. In our future work we plan to examine the effect of such parameters. Moreover, under high computation loads, even highly privacy-sensitive tasks may need to be offloaded. Future research should focus on developing solutions that can better manage the execution of highly privacy-sensitive tasks in such situations.

8 Conclusion

In this paper, we proposed RASH that determines bandwidth and local computation resource allocation in a smart home among multiple competing IoT devices with a goal of minimizing the exposure of privacy-sensitive data while ensuring that tasks can be completed before their delay limit. Given an increasing adoption of Federated Learning (FL) due to its privacy-by-design nature, we considered both *training* tasks and *executing* tasks in our system and their delay, privacy, and computational requirements. Our performance analysis validates the expected performance benefits specifically in preserving privacy by prioritizing higher privacy-sensitive tasks for local execution. The results also highlight the system's effectiveness in task satisfaction while achieving above 90% utilization of local resources. As future directions, we plan to use learning-based algorithms such as reinforcement-based algorithms and include privacy scores in their objective function, jointly optimizing delay, energy, and privacy sensitivity. Our experiments showed training tasks are more often deferred. Therefore, we plan to propose fairness-aware scheduling under overload. Finally, we plan to implement and evaluate our offloading framework as a smart home hub prototype running real applications in a real smart home deployment environment.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10922-025-10022-5.

Author Contributions Rezaei performed all the analysis in the paper. Rezaei and Bayhan contributed in writing the paper. Bayhan also contributed in designing Fig. 2 and the system model. Continella contributed in conceptualizing and designing the privacy assessment metric. Rijswijk-Deij contributed in reviewing the paper and giving feedback on all the analysis. All authors contibuted to the approval of the final version.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Abdellatif, A.A., Mohamed, A., Chiasserini, C.F., Tlili, M., Erbad, A.: Edge computing for smart health: context-aware approaches, opportunities, and challenges. IEEE Netw. **33**(3), 196–203 (2019). https://doi.org/10.1109/MNET.2019.1800083
2. Sovacool, B.K., Del Rio, D.D.F.: Smart home technologies in Europe: a critical review of concepts, benefits, risks and policies. Renew. Sustain. Energy Rev. **120**, 109663 (2020). https://doi.org/10.1016/j.rser.2019.109663
3. Yu, J., Antonio, A., Villalba-Mora, E.: Deep learning (cnn, rnn) applications for smart homes: a systematic review. Computers **11**(2), 26 (2022). https://doi.org/10.3390/computers11020026
4. Zhu, N., Diethe, T., Camplani, M., Tao, L., Burrows, A., Twomey, N., Kaleshi, D., Mirmehdi, M., Flach, P., Craddock, I.: Bridging e-health and the internet of things: The sphere project. IEEE Intell. Syst. **30**(4), 39–46 (2015). https://doi.org/10.1109/MIS.2015.57
5. Zavalyshyn, I., Legay, A., Rath, A., Riviere, E.: Local-first smart home applications with HubOS. In: EDCC 2025: 20th European Dependable Computing Conference (2025)
6. Yuan, D., Ota, K., Dong, M., Zhu, X., Wu, T., Zhang, L., Ma, J.: Intrusion detection for smart home security based on data augmentation with edge computing. In: ICC 2020 - 2020 IEEE International Conference on Communications (ICC), (2020) https://doi.org/10.1109/ICC40277.2020.9148632
7. Liu, Y., Yang, C., Jiang, L., Xie, S., Zhang, Y.: Intelligent edge computing for iot-based energy management in smart cities. IEEE Netw. **33**(2), 111–117 (2019). https://doi.org/10.1109/MNET.2019.1800254
8. Xia, C., Li, W., Chang, X., Delicato, F.C., Yang, T., Zomaya, A.Y.: Edge-based energy management for smart homes. In: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing), pp. 849–856 (2018). https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00-19
9. Ansari, S., Farzaneh, N., Duda, M., Horan, K., Andersson, H.B., Goldberger, Z.D., Nallamothu, B.K., Najarian, K.: A review of automated methods for detection of myocardial ischemia and infarction using electrocardiogram and electronic health records. IEEE Rev. Biomed. Eng. **10**, 264–298 (2017). https://doi.org/10.1109/RBME.2017.2757953
10. Bansal, A., Kumar, S., Bajpai, A., Tiwari, V.N., Nayak, M., Venkatesan, S., Narayanan, R.: Remote health monitoring system for detecting cardiac disorders. IET Syst. Biol. **9**(6), 309–314 (2015). https://doi.org/10.1049/iet-syb.2015.0012
11. Kakria, P., Tripathi, N., Kitipawang, P.: A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors. Int. J. Telemed. Appl. (2015). https://doi.org/10.1155/2015/373474
12. Wu, Q., Chen, X., Zhou, Z., Zhang, J.: Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring. IEEE Trans. Mob. Comput. **21**(8), 2818–2832 (2022). https://doi.org/10.1109/TMC.2020.3045266
13. Han, J., Choi, C.-S., Park, W.-K., Lee, I., Kim, S.-h.: Smart home energy management system including renewable energy based on zigbee and plc. IEEE Trans. Consum. Electron. **60**(2), 198–202 (2014). https://doi.org/10.1109/TCE.2014.6851994

14. Yu, L., Xie, W., Xie, D., Zou, Y., Zhang, D., Sun, Z., Zhang, L., Zhang, Y., Jiang, T.: Deep reinforcement learning for smart home energy management. IEEE Internet Things J. **7**(4), 2751–2762 (2019). https://doi.org/10.1109/JIOT.2019.2957289
15. Taiwo, O., Ezugwu, A.E., Oyelade, O.N., Almutairi, M.S.: Enhanced intelligent smart home control and security system based on deep learning model. Wirel. Commun. Mob. Comput. (2022). https://doi.org/10.1155/2022/9307961
16. Yar, H., Imran, A.S., Khan, Z.A., Sajjad, M., Kastrati, Z.: Towards smart home automation using iot-enabled edge-computing paradigm. Sensors **21**(14), 4932 (2021). https://doi.org/10.3390/s21144932
17. Nguyen, T.D., Marchal, S., Miettinen, M., Fereidooni, H., Asokan, N., Sadeghi, A.-R.: (2019) DÏot: A federated self-learning anomaly detection system for iot. IEEE ICDCS, https://doi.org/10.1109/ICDCS.2019.0008
18. Hafeez, I., Antikainen, M., Ding, A.Y., Tarkoma, S.: Iot-keeper: detecting malicious iot network activity using online traffic analysis at the edge. IEEE Trans. Netw. Serv. Manag. **17**(1), 45–59 (2020). https://doi.org/10.1109/TNSM.2020.2966951
19. Safronov, V., Mandalari, A.M., Dubois, D.J., Choffnes, D., Haddadi, H.: Sunblock: Cloudless protection for iot systems, 322–338 (2024) https://doi.org/10.1007/978-3-031-56252-5_15. Springer
20. Wang, M., Sirlapu, T., Kwasniewska, A., Szankin, M., Bartscherer, M., Nicolas, R.: Speaker recognition using convolutional neural network with minimal training data for smart home solutions. In: 2018 11th International Conference on Human System Interaction (HSI). (2018) https://doi.org/10.1109/HSI.2018.8431363
21. 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). https://doi.org/10.3390/app10165515
22. Huang, B., Li, Y., Li, Z., Pan, L., Wang, S., Xu, Y., Hu, H.: Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing. Future Gener. Comput. Syst. (2019). https://doi.org/10.1155/2019/3816237
23. Elgendy, I.A., Zhang, W., Tian, Y.-C., Li, K.: Resource allocation and computation offloading with data security for mobile edge computing. Future Gener. Comput. Syst. **100**, 531–541 (2019). https://doi.org/10.1016/j.future.2019.05.037
24. Bugeja, J., Davidsson, P., Jacobsson, A.: Functional classification and quantitative analysis of smart connected home devices. In: 2018 Global Internet of Things Summit (GIoTS), pp. 1–6 (2018). https://doi.org/10.1109/GIOTS.2018.8534563. IEEE
25. General Data Protection Regulation (GDPR). https://en.wikipedia.org/wiki/General_Data_Protection_Regulation. Accessed 20 July 2022
26. Kounoudes, A.D., Kapitsaki, G.M.: A mapping of iot user-centric privacy preserving approaches to the gdpr. Internet Things **11**, 100179 (2020). https://doi.org/10.1016/j.iot.2020.100179
27. Chi, H., Zeng, Q., Du, X., Luo, L.: Pfirewall: Semantics-aware customizable data flow control for home automation systems. arXiv preprint arXiv:1910.07987 (2019)
28. Rezaei, T., Bayhan, S., Continella, A., Van Rijswijk-Deij, R.: Erafl: Efficient resource allocation for federated learning training in smart homes. In: NOMS 2024-2024 IEEE Network Operations and Management Symposium, pp. 1–5 (2024). https://doi.org/10.1109/NOMS59830.2024.10575706 . IEEE
29. Gill, S.S., Garraghan, P., Buyya, R.: Router: Fog enabled cloud based intelligent resource management approach for smart home iot devices **154**, 125–138 (2019). https://doi.org/10.1016/j.jss.2019.04.058
30. Liu, H., Li, S., Sun, W.: Resource allocation for edge computing without using cloud center in smart home environment: A pricing approach. Sensors **20**(22), 6545 (2020). https://doi.org/10.3390/s20226545
31. Okafor, O., Esposito, F., Pecorella, T.: Edgeverse: Multi-user virtual reality via edge computing and ebpf. In: 2024 20th International Conference on Network and Service Management (CNSM), pp. 1–4 (2024). https://ieeexplore.ieee.org/document/10814299 . IEEE
32. Shih, Y.-Y., Lin, H.-P., Pang, A.-C., Chuang, C.-C., Chou, C.-T.: An nfv-based service framework for iot applications in edge computing environments. IEEE Trans. Netw. Serv. Manag. **16**(4), 1419–1434 (2019). (https://ieeexplore.ieee.org/document/8880517)
33. Chen, J., Chen, S., Wang, Q., Cao, B., Feng, G., Hu, J.: iraf: a deep reinforcement learning approach for collaborative mobile edge computing iot networks. IEEE Internet Things J. **6**(4), 7011–7024 (2019). https://doi.org/10.1109/JIOT.2019.2913162

34. Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D., Henkel, J.: Computation offloading and resource allocation for low-power iot edge devices. In: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), pp. 7–12 (2016). IEEE
35. Yang, J., Yuan, Q., Chen, S., He, H., Jiang, X., Tan, X.: Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning. IEEE Transactions on Network and Service Management **20**(3), 3205–3219 (2023). (https://ieeexplore.ieee.org/abstract/document/10036357)
36. Papathanail, G., Fotoglou, I., Demertzis, C., Pentelas, A., Sgouromitis, K., Papadimitriou, P., Spatharakis, D., Dimolitsas, I., Dechouniotis, D., Papavassiliou, S.: Cosmos: An orchestration framework for smart computation offloading in edge clouds. In: NOMS, vol. 2020, pp. 1–6. (2020) https://doi.org/10.1109/NOMS47738.2020.9110294
37. Sameri, J., Santos, J., Damme, S.V., Schwarzmann, S., Wei, Q., Trivisonno, R., Turck, F.D., Vega, M.T.: Reinforcement learning-based orchestration of xr applications in distributed 6g cloud infrastructures. IEEE Conference on Network and Service Management (CNSM) (2025)
38. Brahma, J., Sadhya, D.: Preserving contextual privacy for smart home iot devices with dynamic traffic shaping. IEEE Internet Things J. **9**(13), 11434–11441 (2021). https://doi.org/10.1109/JIOT.2021.3126453
39. Bugeja, J., Jacobsson, A., Davidsson, P.: Is your home becoming a spy? a data-centered analysis and classification of smart connected home systems. In: Proceedings of the 10th International Conference on the Internet of Things. IoT '20 (2020). https://doi.org/10.1145/3410992.3411012
40. Bugeja, J., Jacobsson, A., Davidsson, P.: PRASH: a framework for privacy risk analysis of smart homes. Sensors (2021). https://doi.org/10.3390/s21196399
41. Chhetri, C., Genaro Motti, V.: User-centric privacy controls for smart homes. Proc. ACM Hum.-Comput. Interact. **6**(CSCW2), 1–36 (2022). https://doi.org/10.1145/3555769
42. He, X., Liu, J., Jin, R., Dai, H.: Privacy-aware offloading in mobile-edge computing. In: GLOBECOM 2017 - IEEE Global Communications Conference, pp. 1–6 (2017). https://doi.org/10.1109/GLOCOM.2017.8253985
43. Min, M., Wan, X., Xiao, L., Chen, Y., Xia, M., Wu, D., Dai, H.: Learning-based privacy-aware offloading for healthcare iot with energy harvesting. IEEE Internet Things J. **6**(3), 4307–4316 (2019)
44. Wang, Z., Sun, Y., Liu, D., Hu, J., Pang, X., Hu, Y., Ren, K.: Location privacy-aware task offloading in mobile edge computing. IEEE Trans. Mob. Comput. (2023). https://doi.org/10.1109/TMC.2023.3254553
45. Razaq, M.M., Tak, B., Peng, L., Guizani, M.: Privacy-aware collaborative task offloading in fog computing. IEEE Trans. Comput. Soc. Syst. **9**(1), 88–96 (2021). https://doi.org/10.1109/TCSS.2020.3047382
46. Peng, K., Liu, P., Bilal, M., Xu, X., Prezioso, E.: Mobility and privacy-aware offloading of ar applications for healthcare cyber-physical systems in edge computing. IEEE Trans. Netw. Sci. Eng. (2022). https://doi.org/10.1109/TNSE.2022.3185092
47. Singh, A., Auluck, N., Rana, O., Jones, A., Nepal, S.: Scheduling real-time security aware tasks in fog networks. IEEE Trans. Serv. Comput. **14**(6), 1981–1994 (2019). https://doi.org/10.1109/TSC.2019.2914649
48. Reisizadeh, A., Isidoros, T., Hamed, H., Aryan, M., Pedarsani, R.: Straggler-resilient FL: leveraging the interplay between statistical accuracy and system heterogeneity. IEEE J. Sel. Areas Inf. Theory (2022). https://doi.org/10.1109/JSAIT.2022.3205475
49. Ji, Z., Chen, L., Zhao, N., Chen, Y., Wei, G., Yu, F.R.: Computation offloading for edge-assisted federated learning. IEEE Trans. Veh. Technol. **70**(9), 9330–9344 (2021). https://doi.org/10.1109/TVT.2021.3098022
50. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017). https://doi.org/10.1109/MC.2017.9
51. Wang, Y., Wang, L., Zheng, R., Zhao, X., Liu, M.: Latency-optimal computational offloading strategy for sensitive tasks in smart homes. Sensors **21**(7), 2347 (2021). https://doi.org/10.3390/s21072347
52. Wu, D., Yang, W., Zou, X., Xia, W., Li, S., Hu, Z., Zhang, W., Fang, B.: Smart-dnn+: A memory-efficient neural networks compression framework for the model inference. ACM Transactions on Architecture and Code Optimization **20**(4), 1–24 (2023). https://doi.org/10.1145/3498361.3539765
53. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., et al.: Advances and open problems in FL. Foundations and Trends in Machine Learning (2021). https://doi.org/10.1561/2200000083

54. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, B., Van Overveldt, T., Petrou, D., Ramage, D., Roselander, J.: Towards federated learning at scale: System design. In: Talwalkar, A., Smith, V., Zaharia, M. (eds.) Proceedings of Machine Learning and Systems, vol. 1, pp. 374–388 (2019)
55. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., Arcas, B.: Communication-efficient learning of deep networks from decentralized data. In: International Conference on Artificial Intelligence and Statistics (2017)
56. Ye, Y., Li, S., Liu, F., Tang, Y., Hu, W.: Edgefed: Optimized FL based on edge computing. IEEE Access (2020). https://doi.org/10.1109/ACCESS.2020.3038287
57. Zhang, R., Zhang, T., Cai, Z., Li, D., Ma, R.: Memorianova: Optimizing memory-aware model inference for edge computing. ACM Transactions on Architecture and Code Optimization https://doi.org/10.1145/3701997
58. Wess, M., Dinakarrao, S.M.P., Jantsch, A.: Weighted quantization-regularization in dnns for weight memory minimization toward hw implementation. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2929–2939 (2018). https://doi.org/10.1109/TCAD.2018.2857080
59. Park, J.-H., Kim, K.-M., Lee, S.: Quantized sparse training: A unified trainable framework for joint pruning and quantization in dnns. ACM Transactions on Embedded Computing Systems (TECS) **21**(5), 1–22 (2022). https://doi.org/10.1145/3524066
60. Rokh, B., Azarpeyvand, A., Khanteymoori, A.: A comprehensive survey on model quantization for deep neural networks in image classification. ACM Transactions on Intelligent Systems and Technology **14**(6), 1–50 (2023). https://doi.org/10.1145/3623402
61. Gim, I., Ko, J.: Memory-efficient dnn training on mobile devices. In: Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services, pp. 464–476 (2022)
62. Rahmati, I., Shah-Mansouri, H., Movaghar, A.: Qeco: A qoe-oriented computation offloading algorithm based on deep reinforcement learning for mobile edge computing. IEEE Transactions on Network Science and Engineering (2025). https://doi.org/10.1109/TNSE.2025.3556809
63. Wolsey, L.A.: Integer Programming. John Wiley & Sons, Hoboken, NJ, USA (2020)
64. Gurobi techniques. https://www.gurobi.com/resources/mixed-integer-programming-mip-a-primer-on-the-basics/. Accessed: Feburary 2025

Tina Rezaei is a PhD candidate at the University of Twente, Enschede, The Netherlands. She earned her master's degree at Shiraz University, Shiraz, Iran. Her research interests include edge computing and cybersecurity for the Internet of Things.

Suzan Bayhan is currently an Associate Professor with the University of Twente, Enschede, The Netherlands. Previously, she was with the TU Berlin and the University of Helsinki, Helsinki, Finland. Her research interests include resource management in wireless networks, resilience and sustainability of mobile networks.

Andrea Continella is an Associate Professor at the Faculty of Electrical Engineering, Mathematics and Computer Science of the University of Twente, where he leads the cybersecurity team of the Semantics, Cybersecurity & Services group (SCS), and he is a member of the International Secure Systems Lab (iSecLab). His research activity focuses on several aspects of systems security, such as malware and threat analysis, mobile and IoT security, automated vulnerability discovery, and large-scale measurements of security issues. Andrea is a strong advocate for open and reproducible science, he regularly publishes at top-tier security venues, and he serves on the program committees of major systems security conferences and workshops.

Roland van Rijswijk-Deij is full professor of Data-driven Internet Security in the Design and Analysis of Communication Systems group at the Faculty of Electrical Engineering, Mathematics and Computer Science at the University of Twente. His research focuses on analysing the robustness and resilience of Internet infrastructure, studying the impact of new security protocols, and threat intelligence and mitigation, based on large-scale empirical data collection and analysis. He is also scientific director of the Twente University Centre for Cybersecurity Research (TUCCR). Roland previously worked for SURFnet, the national research and education network in The Netherlands and for NLnet Labs, a non-profit organisation that builds open source software for core Internet protocols, such as BGP, RPKI and DNS.

Authors and Affiliations

Tina Rezaei[1] · Suzan Bayhan[1] · Andrea Continella[1] · Roland van Rijswijk-Deij[1]

✉ Tina Rezaei
t.rezaei@utwente.nl

Suzan Bayhan
s.bayhan@utwente.nl

Andrea Continella
a.continella@utwente.nl

Roland van Rijswijk-Deij
r.m.vanrijswijk@utwente.nl

[1] University of Twente, Enschede, The Netherlands