CoRoL: A Reliable Framework for Computation Offloading in Collaborative Robots

Sourabh Bharti, *Member, IEEE*, and Alan McGibney, *Member, IEEE*

***Abstract*—Collaborative robots (cobots) are becoming more prominent in the manufacturing industry due to their ability to operate outside safety zones and work in tandem with humans to perform precise and repetitive tasks, such as visual inspection, product categorization, quality control, etc. Cobots generally have limited computational resources that limit their ability to perform complex machine learning (ML) tasks and as such, various cloud- and fog-based computational task offloading mechanisms have been proposed. However, a growing reluctance to share manufacturing data on the cloud, cybersecurity concerns, and demand of agile decision making is encouraging researchers to design resource-sharing frameworks for on-floor cobots where they can share the execution of complex ML tasks. However, agile on-floor environments and the potential presence of malicious elements make reliable task offloading a significant challenge. This article investigates reliability issues and their effects on executing complex ML task executed by participating on-floor cobots. Specifically, this article aims to answer *whom to offload?* with the objective of ensuring reliability, security, and data protection when offloading computation tasks. To this end, a reputation-based collaborative robotic learning (CoRoL) framework is proposed with the ability to isolate and/or minimize the impact of malicious or poor-performing cobots on computation task execution. In addition, CoRoL is supported by split learning for privacy-preserving task offloading with minimum data exchange. Simulation results and comparative analysis will demonstrate CoRoL's efficiency in terms of percentage of completed tasks, achieved accuracy, and impact on energy consumption.**

***Index Terms*—Collaborative robotics (cobots), dew computing, fog computing, reputation.**

I. Introduction

TRADITIONAL industrial robots are programmed to perform tasks considered to be dangerous for human workers and thus operate behind fences with minimum physical interaction with humans. They are difficult to program and configure and incur a high cost which makes them inaccessible for several small and medium scale organizations. On the other hand, collaborative robots (cobots) [1] for the manufacturing industry are gaining popularity because of their low cost, simple programming, and easy deployment. Cobots work alongside human workers to cooperate in performing precise and repetitive tasks, such as product quality inspection, pick and place [2], etc. Most of the cobots are small in size and can be easily mounted on a workbench, cell, or sturdy cart. Cobots sense and adapt to the environment by gathering data (through sensors) from their surrounding context and process this in real time for agile decision making. Traditionally, real-time data processing on cobots is limited to enable autonomous decisions relating to the speed, distance, proximity, and other variables that ensure the safety of the human worker while performing the designated tasks. However, industrial cobotic applications, such as vision-based product quality inspection [3], require cobots to process high-quality image/video data for inference[1] and accurate decision making. Usually, captured data are offloaded to a cloud server for inference, however, growing concerns over data privacy along with the incurred delay in uploading data to a distant cloud server [4] are prompting organizations to employ a local server or fog node (FN) on the factory floor (Fig. 1). This fog computing [5]-based approach deals with the data privacy and upload delay issues, however, the perils of a single point of failure remains a concern. Another paradigm known as dew computing [6] extends the flexibility of the system as on-floor cobots can collaborate and process a computational task without the support of cloud and fog devices, and in some cases even without an Internet connection. It can make use of centralized nodes such as fog devices as and when required. However, there is no requirement for centralized control in dew computing-based solutions.

A. Motivation

Sophisticated machine learning (ML) frameworks [7] designed to support low latency applications allow cobots to host ML models for local inference. In addition, recent advances in such frameworks also enable resource-constrained devices to locally train the ML model and adjust with unseen input data [8]. Thus, modern cobots can autonomously optimize onboard ML model parameters through real-time training using recently captured input data and make their own decisions without a need of reprogramming. In such scenarios, instead of offloading the model training to a cloud or fog

Manuscript received 16 September 2021; revised 23 November 2021 and 10 January 2022; accepted 26 February 2022. Date of publication 1 March 2022; date of current version 23 September 2022. This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie Grant under Agreement 847577, and in part by the Research Grant from Science Foundation Ireland (SFI) under Grant 16/RC/3918 (Ireland's European Structural and Investment Funds Programmes and the European Regional Development Fund 2014–2020) *(Corresponding author: Sourabh Bharti.)*

The authors are with the Nimbus Research Centre, Munster Technological University, Cork, T12 P928 Ireland (e-mail: sourabh.bharti@mtu.ie; alan.mcgibney@mtu.ie).

Digital Object Identifier 10.1109/JIOT.2022.3155587

[1]Here, *inference* is referred to as processing the captured data using a trained ML model.

device, on-floor cobots can form a resource-sharing ecosystem and combine the available onboard computational resources to collaboratively train their ML models. However, any collaborative ecosystem is susceptible to potential malicious elements working in a selfish manner and sometimes inflicting bad behavior toward other elements. Malicious cobots' behavior can also be influenced by external attackers [9] trying to jeopardize the collaborative model training process [10]. Thus, privacy preservation and reliable offloading[2] are key considerations for a secure and sustainable resource-sharing ecosystem.

B. Research Contributions

Considering the identified challenges, the following contributions are addressed by this work.

1) A framework for collaborative robotic learning (CoRoL) in which co-located cobots on the factory floor can participate in a resource-sharing ecosystem to offload their learning tasks to one another (Fig. 2). CoRoL is driven by the concept of dew robotics and enables on-floor cobots to behave in an autonomous manner with minimum centralized control.
2) As any industrial setup comprises of diverse and heterogeneous devices supplied by different manufacturers, CoRoL takes into consideration the past behavior of cobots while selecting a reliable cobot for offloading (*whom to offload*?). CoRoL also has appropriate provisions to isolate and/or minimize the effects of malicious cobots on task processing.
3) Task offloading in CoRoL is implemented by split learning which enables privacy-preserving offloading without revealing the raw data. In addition, it significantly reduces the amount of offloading data exchange which saves scarce resources such as energy.
4) CoRoL is evaluated based on its effectiveness to detect and eliminate malicious nodes, its impact on ML task's accuracy, ability to reduce the percentage of unsuccessful task execution, and the impact on device resources such as energy.

The remainder of this article is organized as follows. Section II discusses the related research around computational offloading. Section III discusses the system model with problem formulation. A detailed description of CoRoL is given in Section IV along with its components for reliable and privacy-preserving task offloading. Section V discusses the performance evaluation of CoRoL followed by the conclusion and future directions in Section VI.

II. Related Work

An offloading decision making process involves several aspects, such as *when*, *what*, *how*, and *whom to offload*. The decision of *when to offload* is not only governed by the task requirements and current resource availability but also by additional external factors, such as network conditions, cost involved in task offloading, etc. Mechanisms around *what to offload* involve identifying a subset of tasks to be offloaded among a pool of tasks. Common methods used include manual and automatic partitioning of the tasks based on their requirements [11], [12]. Generally, approaches around *how to offload* are classified into system-level, method-level, and application-level offloadings. In the context of modern cobot scenarios and data privacy requirements, this decision is mainly governed by privacy-preserving offloading techniques [13]. This article focuses on the decision of *whom to offload* which involves the selection of a reliable offloading server which can collaboratively execute a complex ML task. In a cobot network, the offloading server can be a cloud server, FN, or nearby cobot. A number of solutions have been proposed in the literature to identify a reliable offloading node, these include approaches focused on: 1) cloud-based; 2) fog-based; and 3) dew-based offloading techniques.

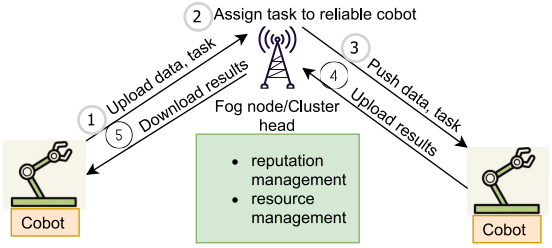

Fig. 1. Typical fog-based cobotic network offloading.

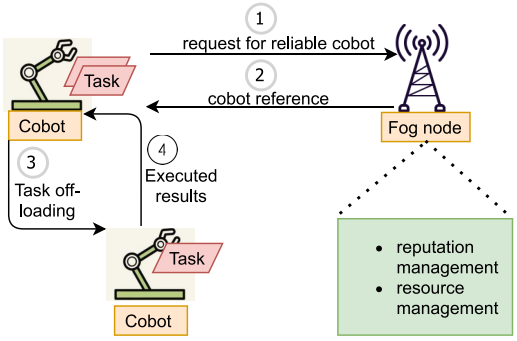

Fig. 2. CoRoL: proposed offloading framework in line with dew computing.

A. Cloud-Based Offloading

Cobots can offload their computational tasks to a cloud server which provides a resource-full computational infrastructure often leveraged in scenarios such as mobile data/task offloading [14]. To this end, various cloud robotic task offloading mechanisms have been proposed for efficient utilization of both computational and communication resources [15]–[18]. Despite several advantages, manufacturing organizations may not necessarily be willing to allow data captured by the cobots to leave the factory floor for cloud-based centralized processing [19]. Reasons include high cost, compromised privacy of commercial intelligence, high communication resource consumption, and hindering agile data processing [20].

[2]Here, *offloading* is referred to as computational offloading among on-floor cobots during model training.

TABLE I
QUALITATIVE COMPARISON OF COROL WITH RECENT OFFLOADING MECHANISMS

Mechanism	Resource-aware	Mobility-aware	Communication	Collaborative model training	Malicious cobots detection	Data privacy handling
[16], 2019	Yes	Yes	Robot-Cloud	No	No	No
[20], 2019	Yes	Yes	Edge-FN	No	No	No
[21], 2020	Yes	Yes	Edge-FN	No	No	No
[23], 2021	Yes	Yes	Cobot-Cobot	No	No	No
CoRoL	Yes	Yes	Cobot-Cobot	Yes	Yes	Yes

B. Fog-Based Offloading

Fog-computing-based task offloading [21], [22] follows the similar process shown in Fig. 1 where the computational task received at a cobot (offloading client) is offloaded to the FN or cluster head (CH) [23] acting as a reliable offloading server at the edge of the network/factory floor. This can help protect an organization's commercial intelligence, reduced task execution time, and cobot energy consumption [5]; as the incurred communication delay and energy spent in transmitting the data to the nearby FN are less when compared with that of a distant cloud data center. However, it brings the perils of a single point of failure where a large number of task requests can overwhelm the FN and induce sufficient delay resulting in an unsuccessful task execution [10]. In this scenario, FN either executes the offloaded task by itself or acts as a coordinator to further offload it to another cobot. However, in either of these cases, complete task information (included training data) is uploaded by the offloading client to FN. In other words, no two cobots can directly interact and collaborate for task processing. Upon task completion, FN uploads the results to the offloading client. Scenarios involving complex computational tasks offloaded to the FN contribute toward the exchange of a large amount of training/model data between cobot and FN; which may result in high resource consumption and delay in task execution. Moreover, the latency in communicating with the FN is still a challenge and can negatively impact in scenarios with scarce local network connectivity [6].

C. Dew-Based Offloading

Dew-based computation [6] offloading fits well for such use case of modern cobots that are expected to independently optimize their onboard ML models for agile decision making. On-floor cobots self-organize and share their onboard computing resources with each other with minimum centralized control or oversight. It works on the concept of microservices designed to support highly distributed applications. Few reliability-based dew offloading mechanisms for cobots have been proposed, such as [23] which contributes toward partially answering the question of *whom to offload?* by considering reputation-based offloading, however, the reputation score is dominated by cobots' computational resource availability without any consideration of the presence of malicious cobots. Similar to the workflow presented in Fig. 1, every offloading instance involves transmitting complete task information to the FN/CH which increases the amount of offloading data exchange in the network and results in increased energy consumption [10]. In addition, the absence of malicious cobots is also assumed at on-floor industrial environments and thus the mechanism has no provision for privacy-preserving offloading. Thus, the aspect to minimize the network resource consumption while maintaining the data privacy still remains unaddressed.

This article considers complex ML model training tasks (Section III, Definition 2) which are computationally too expensive for a single cobot to execute. Thus, the offloading is referred to as complex ML task offloading during model training. As shown in Table I, none of the above-discussed work considers computational offloading during complex ML model training. Recent proposals [24], [25] on exploiting the under-utilized nearby computational resources on edge are focused on appropriate offloading server and task selection to maintain energy-latency tradeoff. However, none of the existing mechanisms consider the presence of nearby malicious nodes and their impact on the overall model training process.

III. SYSTEM COMPONENTS AND PROBLEM DEFINITION

A cobot resource-sharing ecosystem consists of on-floor heterogeneous mobile cobots $C = \{C_1, C_2, C_3, \ldots, C_D\}$ and a unique FN interacting with each other to execute a set of tasks $T(t) = \{T_1(t), T_2(t), \ldots, T_D(t)\}$ in a time period t. It is assumed that a cobot can execute only one task at a time. The mobility of a cobot is attributed to its movement between different processes around the factory floor which may take place several times a day [14]. However, no constant change in its geographical location is expected once the cobot is operating for a particular process. The change in a cobot's geographical location may impact its network connectivity with other cobots [10], hence, its neighboring cobots may vary throughout the day.

Definition 1: C_j is considered as a neighboring cobot of C_i at time t if $\text{mindist}_{C_j \in C}(C_i, C_j) \leq R_{C_i}$ and $\text{mindist}_{C_j \in C}(C_i, C_j) \leq R_{C_j}$, where R_{C_i} and R_{C_j} are the network coverage values of C_i and C_j, respectively.

It is assumed that on-floor neighboring cobots (Definition 1) can communicate with each other and the FN in a single hop manner. FN has a consistent view of the dynamic on-floor robotic network topology and receives periodic updates from the cobots about their available resources. Details on resource models used are given in Section IV-A.

A task is modeled as an ML task where a cobot is to train a particular model with its local data. For example, a vision enabled cobot deployed for product categorization using image processing is expected to train a given ML model so that it

can take autonomous decision about the category of an unseen product.

Definition 2: A cobot C_i's ML task T_i is defined as 4-tuple $\langle C_i$, and $M_i, d_i, \text{EA}_i\rangle$, where C_i is the originating cobot id, M_i is the learning model to be trained using minimum d_i number of data points with the expected accuracy of EA_i.

C_i successfully executes T_i if the corresponding learning model M_i meets the expected accuracy (EA_i) after training. A binary variable S_T is defined in (1) to record the successful/unsuccessful task execution

$$S_{T_i} = \begin{cases} 1, & \text{if } \text{Acc}(M_i) \geq \text{EA}_i \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Cobots can independently choose their task completion strategies in terms of training the ML model locally and/or offloading the model training it to another cobot. In the context of this article, a task offloading step involves an offloading client (cobot willing to offload its ML task) and an offloading server (selected cobot to execute the offloaded task). An ML task offloaded to a reliable offloading server is expected to result in better accuracy as compared to an unreliable offloading server. Here, reliability indicates the cobot's capability to ensure smooth task execution that depends upon available resources, such as computational power, remaining energy, available storage, etc., of the cobot(s) involved in executing the task. However, the unsuccessful task execution can also result due to the presence of malicious cobots acting in a selfish manner (e.g., dropping out in between the training process). Thus, it is important to record the behavior of cobots whenever they are involved in a task execution process. The recorded behavior can be utilized to model their reputation which is leveraged by CoRoL as a decision variable for offloading.

A. Problem Definition

If there are D number of cobots, their offloading strategies in time period t can be defined as $A(t) = \{a_1(t), a_2(t), \ldots, a_D(t)\}$. In a multicobot offloading scenario, the objective is to maximize the successful task executions through selecting reliable offloading servers $C^o(t) = \{C_1^o(t), C_2^o(t), \ldots, C_K^o(t)\} \subset C$, at each time period t. Suppose in a time period t, $T^l(t)$ and $T^o(t)$ represent the set of locally executed and offloaded tasks, respectively, where $\{T^l(t)\} \cup \{T^o(t)\} = T(t)$. Based on (1), the problem of reliable offloading server selection can be defined as follows:

$$\max_{A(t)} \sum_{i=1}^{D} \left[S_{T_i^l}(t) + \sum_{k=1}^{K} S_{T_i^o}(t) \Pr\left[a_i(t) = C_k^o(t)\right] \right] \tag{2}$$

$$\text{s.t. } \forall T_i \in T, C_k^o(t) \in C \tag{3}$$

$$\forall T_i \in T, S_{T_i^l}(t) \geq 0, S_{T_i^o}(t) \geq 0 \tag{4}$$

where $a_i(t) \in C$ is the offloading indicator and $\Pr[a_i(t) = C_k^o(t)]$ is the probability of $a_i(t) = C_k^o(t)$. A task T_i is locally executed if there is no offloading strategy ($a_i(t) = 0$). Otherwise it will be offloaded to the corresponding offloading server. At time t, $S_{T_i^l}(t)$ and $S_{T_i^o}(t)$ record the successful/unsuccessful execution of locally executed and offloaded tasks, respectively. The constraint in (3) makes sure that the offloading servers are among on-floor cobots while the constraint in (4) ensures that the value of (2) never goes below 0 [as S_T is the binary variable defined in (1)]. According to (2), at time t, a task can be offloaded to any of the offloading servers $C_k^o(t) \in C^o(t)$ available on-floor. However, the selection of the specific offloading server is dependent on the offloading strategy (i.e., identifying a reliable offloading server) utilized by the cobot. Thus, the problem addressed in this article is to find out reliable offloading servers so that the value in (2) is maximized.

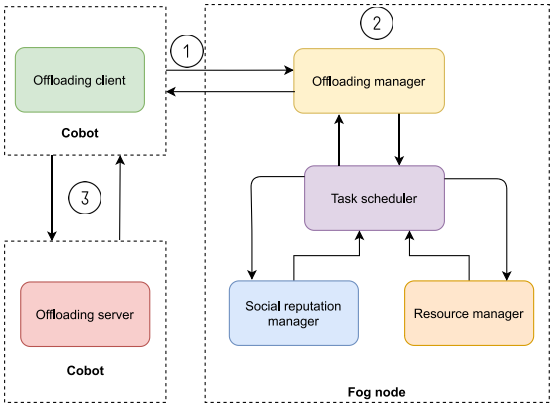

Fig. 3. Logical view of CoRoL.

IV. CoRoL Framework Approach

This section describes the workflow involved in CoRoL for the reliable cobot selection. Fig. 3 shows that offloading enabled by CoRoL involves three distinct entities: 1) an offloading client; 2) offloading server; and 3) local coordinator (FN). The local coordinator can be any fog device at the edge of the local network and mainly consists of offloading manager, task scheduler, resource, and reputation management components. The local coordinator is responsible for receiving requests from offloading clients and scheduling them based on a first-in–first-out approach. It forwards the requests to a task scheduler which acts as a decision support system for reliable cobot selection. The task scheduler consults the resource manager (maintains a record of the available resources of the cobots) and the reputation manager (maintains a record of the reputation) to select a nearby reliable cobot [10].

Upon making the decision the task scheduler communicates the offloading server reference to the offloading manager which returns it to the offloading client to start the task offloading. Upon completion of the task execution, the offloading client records its interaction experience (negative/positive) on the offloading manager to update the reputation record of the offloading server. Computational steps involved in task scheduling are given in Algorithm 1.

Step 1: Offloading client sends a request to the local on-floor FN to return a reliable cobot reference at current time t. The request consists of the task information including learning model as stated in Definition 2: task information along with the originating cobot id are extracted from the task. In the context of CoRoL, a reliable cobot is considered as the one with high social reputation (SR) value (details in Section IV-B).

Algorithm 1 Reliable Cobot Selection at FN

Input : Incoming tasks (Definition 1); Q : task queue
Output: Reliable cobot reference
while *true* **do**
  if $(T_j \leftarrow Q.dequeue()) == \phi$ **then** `// get tasks`
    continue
  end
  `// candidate cobot selection`
  for $i = 1 to |C - 1|$ **do**
    if $(mindist(C_j, C_i) \leq R_{C_j})$ *and* $(mindist(C_j, C_i) \leq R_{C_i})$ *and* $(L_{C_i} < \alpha$ *and* $Re_{C_i} > \beta$ *and* $M_{C_i} > \gamma)$ **then**
      $CC = CC \cup C_i$
    end
  end
  `// reliable cobot selection`
  result = argmax(SR_{C_k}), $\forall C_k \in CC$
  $C_{id} = random$(result)
  return C_{id}
end

Step 2: FN maintains a record of all cobot's resource availability along with their reputation scores in its local database and executes Algorithm 1 involving two subprocesses: first, the FN filters out the neighboring cobots based on the current cobot topology at time t followed by the selection of candidate cobots based on resource availability against task requirements; second, it invokes the reputation function on the candidate cobots to rank the reliable cobot, the one with the highest SR value will be selected. Here, SR (Section IV-B) indicates the reputation value with respect to the on-floor cobot ecosystem. This is followed by the return of the reliable cobot reference to the requesting offloading client.

Step 3: CoRoL utilizes split learning [26] for privacy-preserving task-offloading and execution. Split learning is enabled by splitting the learning model and letting different devices process different components of the model. In the context of CoRoL, the learning model is distributed among the offloading client and the server that consent to collaboratively execute the learning task. The trained model is consolidated by the offloading client at the end of training.

A. Candidate Cobot Selection

As stated in Algorithm 1, the FN filters out neighboring candidate cobots on the basis of their available resources including the following indicators.

1) CPU Load: A reliable cobot should have enough computational power to support the successful execution of a complex ML task which usually involves a lot of arithmetic operations to train the learning model. The CPU load at C_k at time t ($L^t_{C_k}$) is a function of the number of tasks currently being executed by the CPU or in the waiting state [27]. We model the task arrival and processing using $M/G/1$ queue. The length of this queue at time t ($L^t_{C_k}$) represents the current CPU load which is modeled as follows:

$$L^t_{C_k} = \lambda \times \mathrm{ET}^{C_k}_k \tag{5}$$

where, $\lambda = (1/\text{inter} - \text{arrival time})$ and $\mathrm{ET}^{C_k}_k$ (6) are the task T_k's arrival rate and execution time, respectively. $\mathrm{ET}^{C_k}_k$ for T_k executing on a cobot C_k can be estimated as a summation of its waiting time in processing queue (ET^q_k) and the processing time = $[(c_k \times d_k)/f_k]$

$$\mathrm{ET}^{C_k}_k = \mathrm{ET}^q_k + \frac{c_k \times d_k}{f_k} \tag{6}$$

where c_k is the number of CPU cycles required to process d_k data samples and f_k is the CPU cycle frequency of C_k. $c_k = ([O(T_k)]/\delta t)$ is a function of computational complexity of the model used by T_k and the time between consecutive samples (δt) [28]. C_k can also offload T_k to another neighboring cobot C_j. In this case, the total execution time ET^o_k of T_k can be estimated as a sum of local execution time on C_k, offloading data transmission time from C_k to C_j, and local execution time on C_j

$$\mathrm{ET}^o_k = \mathrm{ET}^{C_k}_k + \frac{d(T_k)}{\mu} + \mathrm{ET}^{C_j}_k \tag{7}$$

where $d(T_k)$ is the amount of data transmitted and μ is the data transmission rate. Thus, *ET* depends upon the cobot configuration (CPU cycle frequency), the amount of task data, and the complexity of the learning model.

2) Residual Energy: Since many cobots are battery powered and have limited energy budget, it remains one of the key factors in the process of reliable cobot selection. Every cobot comes with an initial energy level which gets depleted over time as more number of tasks are executed on it. In the context of CoRoL, total energy consumption of a cobot (E^{C_k}) can be estimated as a sum of energy consumed in data exchange during collaborative model training, i.e., network activities ($E^{C_k}_{\mathrm{NET}}$) and energy consumed to process the ML task data ($E^{C_k}_{\mathrm{PRC}}$). Apart from this, the energy consumed in sending periodic resource information updates to the FN is assumed to be negligible

$$E^{C_k} = E^{C_k}_{\mathrm{PRC}} + E^{C_k}_{\mathrm{NET}}. \tag{8}$$

$E^{C_k}_{\mathrm{PRC}}$ is directly proportional to the processing time of the task which depends upon the number of CPU cycles to process one sample of the data (c_k) and CPU cycle frequency (f_k) [29]

$$E^{C_k}_{\mathrm{PRC}} = \mu_k \times \underbrace{\frac{c_k \times d_k}{f_k}}_{\text{processing time}} \tag{9}$$

where μ_k is the constant energy consumption incurred in one time unit of computing chip activity of C_k, and d_k is the number of data samples, respectively.

$E^{C_k}_{\mathrm{NET}}$ can be estimated as the sum of energy consumed in transmission ($E^{C_k}_{\mathrm{Trans}}$) and receiving ($E^{C_k}_{Rcv}$) the data [28], [30]

$$E^{C_k}_{\mathrm{Trans}} = e^k_{\mathrm{amp}} \times r^2 \times B \times k$$
$$E^{C_k}_{Rcv} = e \times B \times k \tag{10}$$

where e^k_{amp} is the energy expenditure for running transmit amplifier, r is the distance to the receiver, B is bandwidth, k is the number of transmission bits, and e is the energy/bit consumed for receiving. If one sample consists of k bits, $E^{C_k}_{\mathrm{Trans}} = (E^{C_k}_{\mathrm{Trans}}/d_k)$ [30].

3) Available Memory: Training of the complex learning model also requires a significant amount of available memory (usually RAM) of the cobot and hence it becomes important to take into account the current available memory for a reliable offloading of the learning task. At time t, a cobot C_k is selected as a candidate cobot if

$$L_{C_k}(t) < \alpha \text{ and } \mathrm{Re}_{C_k}(t) > \beta \text{ and } M_{C_k}(t) > \gamma \tag{11}$$

where $L_{C_k}(t)$ is the CPU load, $\mathrm{Re}_{C_k}(t)$ is the residual energy level, and $M_{C_k}(t)$ is the available memory of C_k at time t. α, β, and γ are the task-specific thresholds for CPU load, residual energy, and available memory for its successful completion.

B. Reliability and Reputation

While processing complex ML tasks, reliability is not only limited to the availability of computational resources or the quality of model training but also mechanisms to deal with the behavior of malicious cobots. One of the behaviors of a malicious cobot is to advertise enough resources to support a successful task execution but always dropping-off during model training. In such case, it is important to record every offloading interaction between cobots as a *positive* or *negative* interaction and utilize this to calculate their reputation scores.

1) *Positive Interaction:* If C_k is selected as the offloading server for C_j and the training process achieves the expected accuracy level within a designated number of training rounds, C_j records a *positive* interaction for C_k.
2) *Negative Interaction:* If C_k is selected as the offloading server for C_j and the training process fails to achieve the expected accuracy level or achieves the expected accuracy but exceeds the designated number of training rounds, C_j records a *negative* interaction for C_k.

The record of both *positive* and *negative* interactions is maintained on the FN in two numeric 2-D arrays $p[D][D]$ and $n[D][D]$, respectively. Entries $p(C_j, C_k)$ and $n(C_j, C_k)$ represent the cumulative *positive* and *negative* interaction values, respectively, reported by C_j during its offloading interactions with C_k.

Based on the past interactions, a cobot C_k's direct reputation for C_j at time t ($\mathrm{DR}^t_{C_j \to C_k}$) can be calculated using the following equation:

$$\mathrm{DR}^t_{C_j \to C_k} = \frac{\sum_{v=1}^{t} p\big(C_j, C_k\big)}{\sum_{v=1}^{t} p\big(C_j, C_k\big) + \sum_{v=1}^{t} n\big(C_j, C_k\big)}. \tag{12}$$

Direct reputation (12) takes both *positive* and *negative* interactions into consideration and is based on the direct interaction between two cobots. However, a malicious cobot can also inflict bad behavior toward other cobots. For instance, the malicious cobot C_j can record a *negative* interaction for C_k even if the interaction was *positive*, to tarnish the reputation of C_k so that the potential resource (C_k) remains unutilized. This behavior can impact in scenarios where the cobots need to satisfy a minimum reputation value (*rth*) in order to be selected as the reliable cobot. In a broader perspective, this can also undermine the performance of the ecosystem as a whole in terms of the number of ML tasks it can successfully execute at a time. To mitigate such attempts, CoRoL considers the indirect reputation of a cobot and models it using subjective logic [31]. This is enabled by FN consulting all other cobots which have interacted with C_k up to time t before taking the offloading decision for C_k. By consultation, we mean taking into consideration the *positive* and *negative* interactions with C_k.

If $\mathbb{C}$ represents the set of cobots interacted with C_k in the past until time t, the SR of C_k, $\mathrm{SR}^t_{C_k}$, in the ecosystem can be estimated as follows:

$$\mathrm{SR}^t_{C_k} = \frac{\sum_{v=1}^{t} \sum_{C_z \in \mathbb{C}} p(C_z, C_k)}{\sum_{v=1}^{t} \sum_{C_z \in \mathbb{C}} p(C_z, C_k) + \sum_{v=1}^{t} \sum_{C_z \in \mathbb{C}} n(C_z, C_k)}. \tag{13}$$

As stated in Algorithm 1, if a cobot C_j is to offload a learning task to another cobot, FN ranks the candidate cobots in increasing order of their SR values. The candidate cobot with the highest SR value is selected as the most reliable offloading server for C_j.

In scenarios where more than one candidate cobot has the same SR value, the offloading server is selected randomly among them. The SR value of a malicious cobot (dropping-off in between task executions) remains low as other candidate cobots (with higher SR values) are given preference for the reliable offloading server selection. As a result of which the malicious cobot is selected only if there is no other candidate cobot and its SR value is greater than the *rth* value specified by the offloading client. This restricts the cobots ability to restore their SR overtime. In scenarios with the high *rth* values, such malicious cobot are never selected which may result in underutilization of the on-floor resources in case such cobot can revert its behavior to a healthy cobot. To deal with such a scenario, FN randomly (e.g., once a day) offloads a dummy ML task to the cobot(s) with the SR values below 0.5 (initial reputation value) to assess whether they have changed their behavior. If the task is executed successfully, FN records a positive interaction for the cobot; which helps such cobots to push their SR values beyond *rth* in order to get selected as offloading servers. This not only detects the (malicious) cobots with changed behavior but also helps other healthy cobots whose SR value was decreased (below 0.5) due to *negative* interaction reported by a bad-mouthing cobot. Such assessment by FN causes minimal overhead to the CoRoL without disrupting the normal task executions.

C. ML Task Offloading With Minimum Data Exchange

CoRoL utilizes split learning [26] to enable privacy-preserving offloading while executing the task. Split learning utilizes artificial neural networks (ANNs) and is driven by distributing the ANN layers among multiple devices (cobots in our case) that consent to aggregate the model at the end of training. A simple split learning architecture (Fig. 4) can be realized using two cobots (C_j and C_k) where C_j is executing the front end ($\mathrm{layer}_1, \mathrm{layer}_2, \ldots, \mathrm{layer}_m$) of the neural network and shares the activations and gradients corresponding to layer_m with C_k. C_k executes the back end ($\mathrm{layer}_{m+1}, \mathrm{layer}_{m+2}, \ldots, \mathrm{layer}_n$) of the neural network and

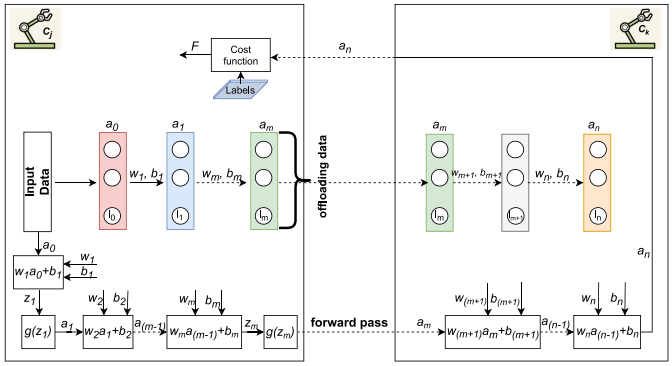

Fig. 4. ANN model offloading in CoRoL.

sends back the output$_n$ to the C_j where the cost function, F(output$_n$, labels) is executed.

Here, layer$_m$ is also known as "split layer." The process continues until the model converges. C_j and C_k are learning the same model in a collaborative manner without revealing the training data. It is to be noted that split learning also enables C_j to retain the labels which further strengthens the raw data privacy aspect. Backpropagation is initiated by C_j computing σ (14) and sending it to C_k

$$\sigma = \frac{\partial F}{\partial a_n}. \tag{14}$$

C_k calculates the partial derivatives as shown as follows:

$$\begin{aligned} \frac{\partial F}{\partial a_{n-1}} &= \sigma . \frac{\partial a_n}{\partial z_n} . \frac{\partial z_n}{\partial a_{n-1}} \\ &\cdots \\ \frac{\partial F}{\partial a_m} &= \frac{\partial F}{\partial a_{m+1}} . \frac{\partial a_{m+1}}{\partial z_{m+1}} . \frac{\partial z_{m+1}}{\partial a_m}. \end{aligned} \tag{15}$$

C_k shares $(\partial F/\partial a_m)$ with C_j in order to complete the backpropagation. C_j calculates the remaining partial derivatives as shown in (16). This process continues until the model converges

$$\begin{aligned} \frac{\partial F}{\partial a_{m-1}} &= \frac{\partial F}{\partial a_m} . \frac{\partial a_m}{\partial z_m} . \frac{\partial z_m}{\partial a_{m-1}} \\ &\cdots \\ \frac{\partial F}{\partial w_1} &= \frac{\partial F}{\partial a_1} . \frac{\partial a_1}{\partial z_1} . \frac{\partial z_1}{\partial w_1}. \end{aligned} \tag{16}$$

On ensuring the privacy of offloading data, if C_j is to offload its ML task to C_k, it does not need to transmit the training data and corresponding labels to C_k. Rather the offloading data transmission from C_j to C_k is limited to the number of activations in the split layer (Fig. 4). During the training, only the gradient outputs corresponding to the split layer are exchanged between C_j and C_k. This significantly reduces the amount of data exchange between the offloading client and the server which results in reduced energy consumption. Split-learning-based offloading also gives the flexibility as to how many layers an offloading client has to offload based on the amount of resources it possesses. However, the decision on the number of offloading layers (*when to offload?*) is not in the scope of this work.

1) Reliability and Performance Tradeoff: Although CoRoL brings the benefits of reliable computation offloading, it is important to observe its effect on system performance as reliability comes with an additional cost [32], [33]. To this end, a tradeoff metric (τ) is defined as follows:

$$\tau = \pi \times X \tag{17}$$

where X is the reliability metric represented as the system throughput in the presence of CoRoL. In other words, X can be measured by the ratio of the number of successful task executions (1) against the total number of tasks. On the other hand, π represents the cost involved, which can be measured by analyzing the computational overhead and the amount of data exchange during offloading. As CoRoL attempts to maximize X by providing a reputation-based offloading server selection, it also minimizes π by minimizing the computational and communication complexities as discussed as follows.

Computational Complexity: The offloading is governed by split learning which involves partitioning the ANN layers and distributing them to multiple cobots for training. If each layer is represented as a matrix, the computational complexity contributed by propagating from layer$_0$ to layer$_1$ can be calculated as $O(l_0 * l_1 * z)$, where l_0 and l_1 are the number of neurons in layer$_0$ and layer$_1$, respectively, and z represents the number of training examples. This is followed by activation operation which contributes to the complexity of $O(l_1 * z)$. Thus, total complexity contributed by a feedforward from layer$_0$ to layer$_1$ is: $O(l_1 * z + l_0 l_1 * z) = O(l_0 l_1 * z)$. The computational complexity incurred by backpropagation is same as feedforward. Thus, the total complexity with two layers for one epoch is $O(l_0 l_1 * z)$. According to Fig. 4, ANN layers at C_j and C_k contribute to the computational complexity of $O(z * (l_0 l_1 + l_1 l_2 + \cdots + l_{m-1} l_m))$ and $O(z * (l_m l_{m+1} + l_{m+1} l_{m+2} + \cdots + l_{n-1} l_n))$, respectively, for one epoch.

Communication Complexity: As for data exchange between the offloading client and the server, C_j shares the activations corresponding to the split layer with C_k (Fig. 4) for feedforward and the same number of activations are shared by C_k with C_j during backpropagation. After the training finishes, C_k shares the trained model weights with C_j for trained model integration. To this end, if N represents the number of model parameters and l_s as the size of split layer then the total communication complexity can be estimated as follows:

$$\begin{aligned} \text{Communication per epoch} = &\underbrace{z * l_s}_{\text{feedforward}} + \underbrace{z * l_s}_{\text{backpropagation}} + \eta N = \\ &= O(2 * z * l_s + \eta N) \end{aligned} \tag{18}$$

where, η is the fraction of model parameters C_k shares with C_j at the end of the training.

V. Performance Evaluation

A. Experiment Setup and Scenarios

The factory-floor scenario is simulated using the PySyft [34] environment where cobots are virtual workers interacting each other to execute ML tasks. A total of ten cobots and one FN are considered for experiments in which two are malicious

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Factory-floor area	$50m \times 50m$
Number of cobots	10
Number of tasks in a day	20
Data-set	MNIST
Total number of samples	$60,000$
Expected accuracy	97%
Predictive model	MLP
Mobility model	Random waypoint
Cobot transmission range	$25m$
Inter-arrival time of tasks	10 time units
Initial energy of cobots	1000 J
Initial memory	1000MB
Initial reputation value	0.5
B, CPU cycle frequency	1bps, 1.8GHz
$e_{amp} = e, r$	$50\times10^{-9}J/bit/m^2$,0-25m
α, β, γ	2, [250], [100]

cobots and eight are nonmalicious. For representation purpose, malicious cobots are termed as follows.

1) *MCobot1:* Always advertises enough resources to complete the task but drops-off in between task execution.
2) *MCobot2:* Launches bad mouthing (reports negative interaction) whenever interacts with other cobots.

The reputation database is populated over 15 days in which 20 tasks were executed every day. Every cobot can only perform a maximum of two tasks in a day. The benchmark MNIST data set is used to train the task models and the expected test accuracy for the successful completion of the task is kept as 97%. A lightweight MLP model is considered as the task model with the following architecture: (784∗128)(128∗640)(640∗10). For simulation purpose, the training is performed on a device with Intel core i7 processor (2.3 GHz), 32 GB RAM, and 256-GB SSD. Sixty thousand images were used for training whereas 10 000 were used for testing purpose. Table II lists the key simulation parameters used during the course of the experiments. Each task is considered to be a model training task for the MNIST data set using 60 000 training images. Given the number of cobots in the ecosystem and the number of tasks per day (Table I), algorithm parameters (α, β, γ) were calculated using (5)–(10). The values of α are calculated as 250 J, whereas the value of β is calculated in terms of a maximum number of tasks a cobot can execute in a day (kept as 2). The value of γ (100 MB) is adopted from the empirical results presented in [13] about the memory expenditure in split learning. Another algorithm parameter, i.e., cobot's transmission range's value (25 m) is assumed to be half of the factory-floor area.

This section demonstrates the evolution of Algorithm 1 and the process of reliable offloading server selection. It also explores the utility of CoRoL in terms of its ability to detect *MCobot1* and prevent its participation toward the task execution in the on-floor ecosystem. To show CoRoL's effect on minimizing the impact of *MCobot2*, we consider applications

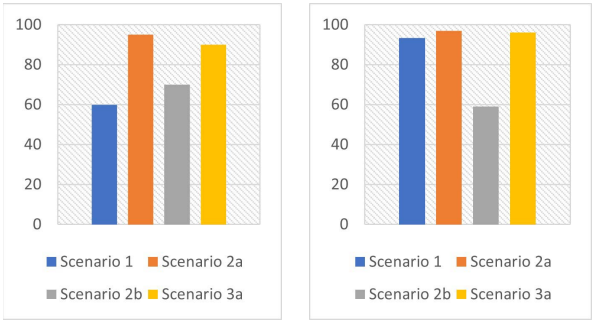

Fig. 5. Performance of different scenarios in presence of *MCobot1* (number of tasks = 140, simulation duration = 7 days, and *rth* = 0.5 for Scenario 3a). (a) Percentage of successful task executions. (b) Average task accuracy.

with different reputation threshold requirements (as discussed in Section IV-B). The following scenarios are defined for evaluation purposes.

Scenario 1: Represents the baseline scenario for random cobot selection without the presence of malicious cobots.

Scenario 2: This scenario considers first filtering out the neighboring candidate cobots on the basis of their available resources to execute the task followed by randomly selecting one of those candidate cobots.

1) *Scenario 2a (With No Malicious Cobot):* Scenario commonly used in the literature to enable networked robotics.
2) *Scenario 2b:* With the presence of *MCobot1*.

Scenario 3: This scenario represents CoRoL with different operational settings and reputation threshold (*rth*) values.

1) *Scenario 3a:* Direct-reputation-based cobot selection with *MCobot1*.
2) *Scenario 3b:* Direct-reputation-based cobot selection with *MCobot1* and *MCobot2*.
3) *Scenario 3c:* SR-based cobot selection (Algorithm 1) with *MCobot1* and *MCobot2*.

B. Results and Analysis

1) Effect of MCobot1 and CoRoL Response: Fig. 5 shows the results obtained by the experiments performed across the different scenarios. Scenario 1 results in the least number of successfully executed tasks (60%) many of the them were assigned to the cobots without sufficient resources available because of the random selection approach used.

The performance improves with Scenario 2a (97% accuracy and 94% successful task execution) when the offloading decision is taken on the basis of distance and available resources, and thus results support the rationale for its incorporation into offloading decision. However, in the presence of *MCobot1*, Scenario 2b suffers on both accuracy (59%) and the percentage of successfully executed tasks (70%) as *MCobot1* always advertises enough resources to execute the task.

Fig. 6 shows Scenario 3a and the DR value progression for *MCobot1* when it starts dropping-off in between task execution from day 2 of the simulation, compared with that of the reliable cobot.

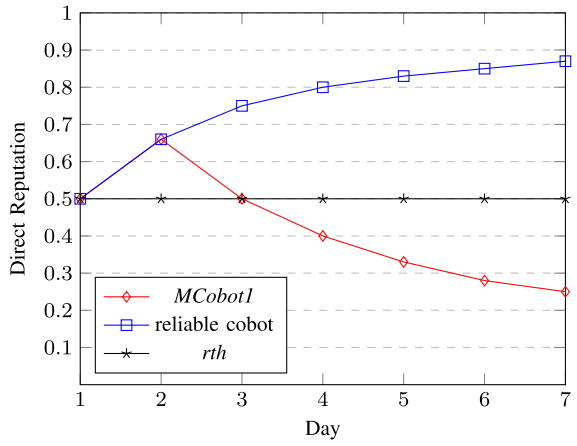

Fig. 6. Direct reputation value progress for *MCobot1* and reliable cobot.

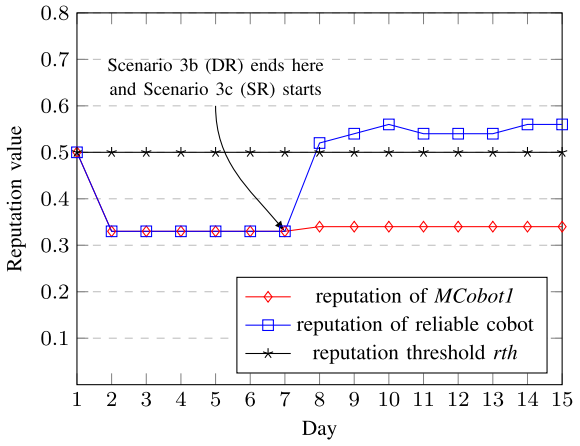

Fig. 8. CoRoL's impact on reputation values of reliable cobot and *MCobot1* in the presence of *MCobot2*.

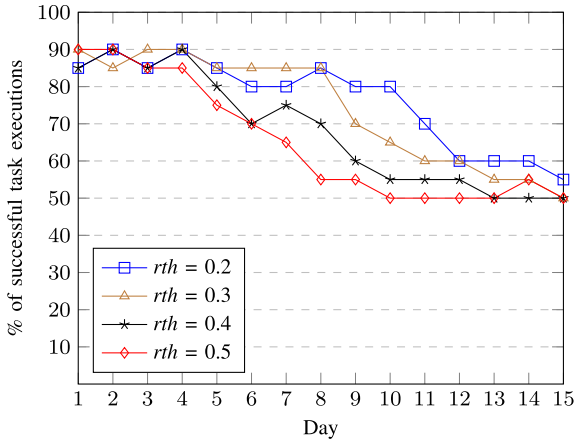

Fig. 7. Effect of different *rth* values on Scenario 3b with one *MCobot2*.

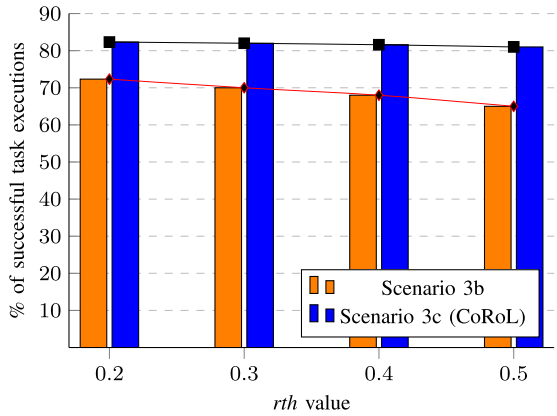

Fig. 9. Comparison between Scenarios 3b and 3c (days = 15 and number of tasks >200).

Low DR value for *MCobot1* prevents its participation in the ecosystem which results in improved test accuracy (96.1%) and increased percentage of successfully executed tasks (90%) in Scenario 3a (Fig. 5).

2) Effect of MCobot2 and CoRoL Response: Scenario 3a deals with the *MCobot1* (Figs. 5 and 6) by identifying it based on its low DR value, however, *MCobot2* can still launch a bad-mouthing attack and degrade the performance of the overall system in terms of the number of successful task executions. The effect of *MCobot2* can be observed in an application which requires a reliable cobot to satisfy a reputation threshold value (*rth*) in order to be selected as the offloading server. As *MCobot2* records negative interaction every time it interacts with other cobots, i.e., bad mouthing, its DR values get smaller over time and fewer options are available for the selection of offloading server in the presence of a larger *rth*. This is evident from Fig. 7 as when Scenario 3b is exposed to *MCobot2*, the percentage of successfully executed tasks varied with different *rth* values. The effect of *MCobot2* is observed early (day 7) with high *rth* value (0.5). On the other hand, low *rth* values (0.2, 0.3) exhibit better performance as the reduced DR values of other cobots due to *MCobot2* are still greater than *rth* and thus satisfy the criteria to be selected as offloading servers.

To minimize the impact of *MCobot2*, CoRoL not only considers DR values but also the SR [enabled by (13)] at the time of the reliable cobot selection. Fig. 8 shows *MCobot2*'s reputation value comparison for a cobot under attack in Scenarios 3b and 3c with *rth* value of 0.5. Scenario 3b considers only DR values for reliable cobot selection. As soon as *MCobot2* records a negative interaction, its DR value for the cobot under attack is decreased to 0.33 (day 2) which remains constant till day 7 as $rth = 0.5(>0.33)$ prevents the cobot under attack to become the offloading server for *MCobot2*. However, for this duration (days 1–7), the cobot under attack continued to be selected as the offloading server for other cobots. Due to this, when CoRoL (Scenario 3c) takes over from day 8, the indirect reputation value of the cobot under attack improves its SR in the ecosystem (Fig. 8) and pushes it beyond the *rth* value of 0.5 to be selected as the offloading server.

However, it is to be noted that the reputation value of *MCobot1* remains low even after consideration of indirect reputation. This shows that CoRoL can not only detect *MCobot1* and prevent it from participating in the task execution process but also minimizes the effect of *MCobot2* on a reliable cobot. The impact of CoRoL is manifested in Fig. 9 as Scenario 3c (CoRoL) improves the performance of DR-based Scenario 3b

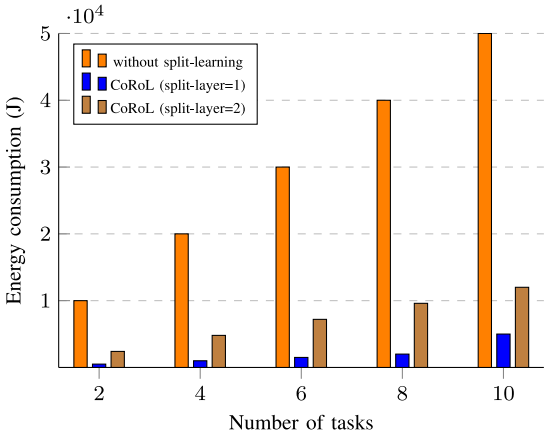

Fig. 10. Energy saving benefits of split-learning-based computation offloading.

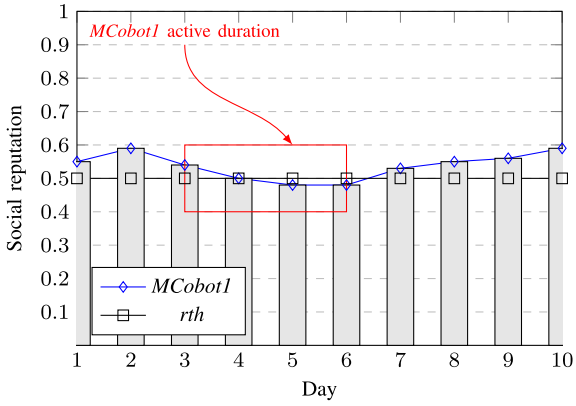

Fig. 11. SR value progression for *MCobot1* with changeable behavior.

in terms of the percentage of the successfully executed tasks. For each *rth* value, extensive simulations were conducted and the values in Fig. 9 represent the average of the values obtained over 15 days of task execution. Scenario 3b clearly shows a decline trend in the percentage of successful executions with increasing the *rth* value when exposed to both *MCobot1* and *MCobot2*. On the contrary, Scenario 3c exhibits the stability irrespective of the *rth* value. As the average task accuracy is proportional to the number of successful executions, Scenario 3c also shows higher average task accuracy as compared to Scenario 3b.

3) Effect of CoRoL on Energy Consumption: This section discusses the benefits of utilizing split-learning-based offloading on energy consumption and performs the comparative analysis with recent dew-based offloading mechanism [23]. The result of task execution is trained model parameters to be utilized by the offloading client for the prediction. For instance, the learning model used for experiments in this work comprised of 188672 ($= 784 * 128 + 128 * 640 + 640 * 10$) weight parameters. Existing dew-based offloading mechanisms [6], [23] involve model parameters to be shared between: 1) offloading client→FN/CH; 2) FN/CH→offloading server; 3) FN/CH←offloading server; and 4) offloading client←FN/CH. offloading such ML tasks in full involves transmitting corresponding model parameters along with the input data, thus increases the energy consumption [10]. To reduce this model parameter data exchange, CoRoL limits the FN's usage to the reliable offloading server selection and reputation management. This is followed by the direct interaction of the offloading client and the server for task execution using split learning. As discussed in Section IV-C, split learning eliminates the need to share raw data and complete model parameters with the offloading server. Instead, the data exchange between the offloading client and the server while task execution is limited to the size of the split layer. Finally, the offloading server transmits the model parameters corresponding to the offloaded layers to the client when the training process terminates. Fig. 10 shows the numerical results corresponding to overall energy consumed by split learning (used in CoRoL) and without split-learning-based offloading during the execution of ML tasks. The image data size for [23] is kept as ($32 \times 32 \times 3$). The energy consumption in [23] is contributed by model size and training data of the task, on the other hand, CoRoL prevents transmitting training data and also reduces the offloaded model data. To test the robustness of CoRoL, we performed the experiments with different split layer indices (=1 & 2) for the used MLP model as the amount of offloaded model data varies with the split-layer index value. Thus, CoRoL not only minimizes the adverse effects of the malicious elements in the ecosystem but also helps cobots minimizing their energy consumption in the task execution process.

4) Cobots With Changeable Behavior and CoRoL Response: This section discusses CoRoL's response to *MCobot1* and *MCobot2* with uncertain or changeable behavior. The frequency of periodic check performed by FN for behavior change (Section VI-B) of malicious cobots is kept as once in a day for this experiment. Fig. 11 shows the results corresponding to a scenarios where *MCobot1* is active[3] from day 3 to day 6 and is inactive on all other days. As shown in Fig. 11, the SR value of *MCobot1* decreases as it starts dropping-off in between task executions (i.e., becomes active). However, when it wants to change its behavior, it is observed by FN as a result of its periodic checks and a positive response is registered for *MCobot1* at the end of day 6 which pushes its SR value above *rth*. This enables inactive *MCobot1* to be selected as an offloading server as can be observed from its increased SR value when it is inactive.

Fig. 12 shows the results corresponding to a scenario where *MCobot2* is active from day 1 to day 5 and changes its behavior on day 6 till day 7; further it becomes active again on day 8 till day 10. During day 1 to day 10, *MCobot2* interacts with Cobot1, Cobot2, Cobot5, Cobot7, Cobot8, Cobot9, and Cobot10. Thus, we observe the SR values of these cobots (in the presence of CoRoL) throughout this duration as the objective of an active *MCobot2* is to record negative interaction for the other cobots it interacts with so that their SR value is decreased and falls below *rth*. This decreases the overall system throughput as the resources (i.e., attacked cobots)

[3]Here, "active" and "inactive" are referred to as when a cobot behaves like a malicious cobot and healthy cobot, respectively.

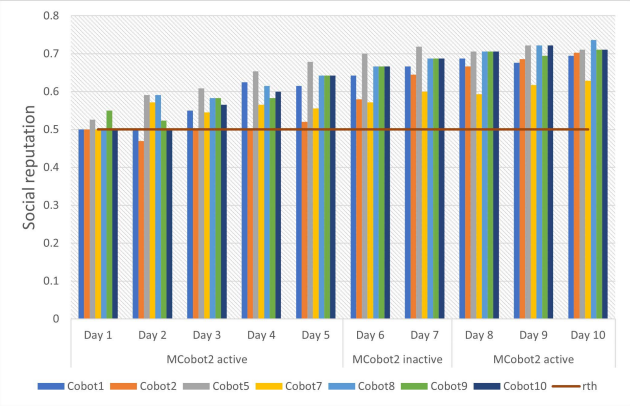

Fig. 12. Effect of *MCobot2* on SR values of cobots under attack.

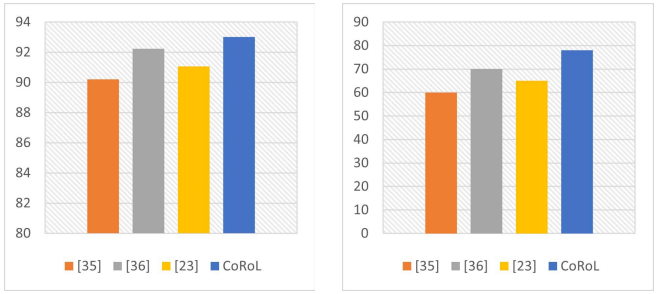

Fig. 13. Comparative evaluation of CoRoL with state of the art. (a) Average task accuracy. (b) Percentage of successful task executions.

remain underutilized. The *rth* value for this experiment was kept as 0.5. As shown in Fig. 12, on day 2, the SR value of Cobot2 goes below *rth* because of *MCobot2* as it was observed that none of the unsuccessful task execution involved Cobot2 on that day. However, periodic checks from FN for cobot(s) with lowest SR value push the SR value of Cobot2 beyond *r*th, i.e., restoring it reputation. Apart from Cobot2, none of the attacked cobots' SR value is less than *rth* during the active and inactive periods of *MCobot2* as its effect is minimized by taking into consideration the SR value instead of DR. SR includes the recommendation (13) from other healthy cobots also which minimizes the effect of a bad-mouthing cobot, i.e., active *MCobot2*.

The above experiments show how CoRoL can facilitate behavioral changes of cobots over time and support them to restore their SR values' in a manner that does minimizes the risk to task execution (i.e., use of dummy task).

5) Comparative Analysis: This section discusses the comparative evaluation of CoRoL with state-of-the-art offloading mechanisms proposed in [23], [35], and [36]. The mechanism proposed in [35] works on the principle of uniform selection probability of the offloading server and mainly focuses on the decision of whether to offload the task on a local FN or not by comparing the energy and time expenditures for available FNs (i.e., resource aware), whereas [36] selects the nearest available device as the offloading server (i.e., distance based) to minimize the response time. In addition to these two approaches, CoRoL is also compared with a reward-based offloading server selection mechanism [23]. The mechanism proposed in [23] works on the principle of rewarding an offloading server (robot) upon a successful task execution and recording the reward values for future offloading server selection. As CoRoL is focused on executing ML tasks, we considered the definition of successful task execution for [23], the same as given in (1). A cluster of on-floor cobots with a CH is implemented to reproduce [23] for fair comparison with CoRoL. Results presented in this section correspond to the scenarios where both type of malicious cobots, i.e., *MCobot1* and *MCobot2* are active in the ecosystem. Since the proposal deals in ML tasks, we choose: 1) average task accuracy and 2) percentage of successful task execution, as the evaluation parameters for this comparative evaluation. For fair comparative evaluation, we implemented split-learning-based model training for [23], [35], and [36], although the decision of offloading server selection was based on their selection mechanisms. Results shown in Fig. 13 correspond to the simulation duration of 7 days with more than 100 tasks executed in that duration. The *rth* value for the CoRoL scenario is kept as 0.5.

CoRoL outperforms [23], [35], and [36] in terms of average task accuracy and percentage of tasks successfully executed for the simulation duration. Although both [35] and [36] are resource aware but have no provision for identifying the malicious cobots which can be selected as offloading servers if not isolated. On the other hand, [23] is able to deal with *MCobot1* as the offloading server selection is reward based, however, it is not resource aware and does not have any provision to mitigate the effects of *MCobot2* on the overall task execution process. Whereas CoRoL is not only resource aware but also considers both direct and SR to mitigate the effects of *MCobot1* and *MCobot2*, respectively. From computational complexity point of view, each of the mechanisms: [23], [35], [36], and CoRoL, sort the devices on the basis of their available resources (energy and time), distance, reward, and SR values, respectively. If C represents the set of potential offloading servers, all mechanisms exhibit the computational complexity of $O(|C|\log|C|)$ for the reliable offloading server selection. However, since CoRoL also supports collaborative model training using split learning (Section IV-C1), the overall computational complexity can be estimated as follows:

$$\underbrace{O(|C|\log|C|)}_{\text{Offloading server selection}} + \underbrace{\begin{array}{l} O(z*(l_0l_1+l_1l_2+\cdots+l_{m-1}l_m)) \\ +\, O(z*(l_ml_{m+1}+l_{m+1}l_{m+2}+\cdots+l_{n-1}l_n)) \end{array}}_{\text{Model training}}. \quad (19)$$

VI. Conclusion and Future Work

This article presented a framework for on-floor cobots resource sharing to support complex ML tasks' execution in a collaborative fashion across an cobot ecosystem. CoRoL focused on supporting resource efficiency as well as managing the potential presence of malicious entities, in a manufacturing setup with heterogeneous devices and potentially

safety-critical application scenarios. Thus, the identification and provision to minimize the impact of malicious or underperforming cobots on ML task processing are essential for the sustainability and effectiveness of such on-floor ecosystem. The reputation-based reliable offloading in CoRoL enables a context-driven cobot selection by taking into account both available resources and reputation values of the cobots. Split learning is utilized to implement offloading among cobots. Simulation results indicate that CoRoL successfully detects and isolates malicious cobots and minimizes their impact on computation tasks. In addition, it significantly reduces the energy consumption in computation task processing when compared to the state of the art.

Future work will involve the implementation and deployment of the CoRoL approach under real world conditions (physical testbed) to analyze its behavior during both stringent and flexible network conditions. In addition, functionalities of CoRoL can also be extended by troubleshooting malicious cobots with certain bad behavior. The benefits of CoRoL can map to existing scenarios currently being developed by industry. For example, automated visual inspectors developed by IBM in collaboration with Apple [37] utilize the CoreML framework [7] to deploy a predictive model trained for product quality inspection and damage detection on iOS devices with multiple connectivity options, such as 5G and Bluetooth. These visual inspectors are mobile and easy to install when compared with stationary cameras. Although CoreML provides the flexibility of training the model on-device, current solutions are limited to on-device inference. CoRoL can be leveraged to enable on-floor visual inspectors form an ecosystem and share their computational resources to execute an ML task. The use of a reputation-based offloading server selection mechanism as proposed in CoRoL could ensure reliable ML task execution, while also allowing the use of the heterogeneous on-floor inspector infrastructure (i.e., different types, makes, cost, and functionalities).

Another consideration is answering the question of *when to offload* based on the current state of onboard and network resources. In the context of CoRoL, this will involve identifying the optimum number of neural network layers required to be offloaded in the presence of resource constraints.

REFERENCES

[1] S. Kianoush, S. Savazzi, M. Beschi, S. Sigg, and V. Rampa, "A multisensory edge-cloud platform for opportunistic radio sensing in cobot environments," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 1154–1168, Jan. 2021.

[2] E. Matheson, R. Minto, E. G. G. Zampieri, M. Faccio, and G. Rosati, "Human–robot collaboration in manufacturing applications: A review," *Robotics*, vol. 8, no. 4, p. 100, 2019.

[3] M. L. H. Souza, C. A. da Costa, G. de Oliveira Ramos, and R. da Rosa Righi, "A survey on decision-making based on system reliability in the context of industry 4.0," *J. Manuf. Syst.*, vol. 56, pp. 133–156, Jul. 2020.

[4] I. A. Elgendy and R. Yadav, *Survey on Mobile Edge-Cloud Computing: A Taxonomy on Computation Offloading Approaches*. Cham, Switzerland: Springer Int., 2022, pp. 117–158.

[5] A. Kattepur, H. Dohare, V. Mushunuri, H. K. Rath, and A. Simha, "Resource constrained offloading in fog computing," in *Proc. 1st Workshop Middlew. Edge Clouds Cloudlets*, 2016, pp. 1–6.

[6] A. Botta, L. Gallo, and G. Ventre, "Cloud, fog, and dew robotics: Architectures for next generation applications," in *Proc. 7th IEEE Int. Conf. Mobile Cloud Comput. Serv. Eng. (MobileCloud)*, Newark, CA, USA, 2019, pp. 16–23.

[7] "Core ML: Integrate Machine Learning Models Into Your App." [Online]. Available: https://developer.apple.com/documentation/coreml (Accessed: Jul. 7, 2021).

[8] "Collaborative Robotics:Cobots are Collaborators. A.I. Will Make Them Partners." [Online]. Available: https://www.mouser.ie/applications/collaborative-robotics-ai-make-partners/ (Accessed: Jul. 7, 2021).

[9] J.-P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, "Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations," *Int. J. Inf. Security*, vol. 21, pp. 115–158, Mar. 2021.

[10] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 842–870, 2nd Quart., 2021.

[11] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, 2012, pp. 945–953.

[12] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.

[13] S. Bharti and A. Mcgibney, "Privacy-aware resource sharing in cross-device federated model training for collaborative predictive maintenance," *IEEE Access*, vol. 9, pp. 120367–120379, 2021.

[14] "How to Decide If You Need a Mobile Cobot." 2018. [Online]. Available: https://blog.robotiq.com/how-to-decide-if-you-need-a-mobile-cobot

[15] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: Architecture, challenges and applications," *IEEE Netw.*, vol. 26, no. 3, pp. 21–28, May/Jun. 2012.

[16] A. Rahman, J. Jin, A. L. Cricenti, A. Rahman, and A. Kulkarni, "Communication-aware cloud robotic task offloading with on-demand mobility for smart factory maintenance," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 2500–2511, May 2019.

[17] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015.

[18] A. Rahman, J. Jin, A. Cricenti, A. Rahman, and D. Yuan, "A cloud robotics framework of optimal task offloading for smart city applications," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Washington, DC, USA, 2016, pp. 1–7.

[19] M. Dhada, A. K. Jain, M. Herrera, M. P. Hernandez, and A. K. Parlikad, "Secure and communications-efficient collaborative prognosis," *IET Collab. Intell. Manuf.*, vol. 2, no. 4, pp. 164–173, 2020.

[20] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4225–4234, Jul. 2019.

[21] R. Roshan, R. Matam, M. Mukherjee, J. Lloret, and S. Tripathy, "A secure task-offloading framework for cooperative fog computing environment," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Taipei, Taiwan, 2020, pp. 1–6.

[22] M. Al-Khafajiy *et al.*, "COMITMENT: A fog computing trust management approach," *J. Parallel Distrib. Comput.*, vol. 137, pp. 1–16, Sep. 2021.

[23] A. W. Malik, A. U. Rahman, M. Ali, and M. M. Santos, "Symbiotic robotics network for efficient task offloading in smart industry," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4594–4601, Jul. 2021.

[24] R. Yadav, W. Zhang, O. Kaiwartya, H. Song, and S. Yu, "Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14198–14211, Dec. 2020.

[25] R. Yadav *et al.*, "Smart healthcare: RL-based task offloading scheme for edge-enable sensor networks," *IEEE Sensors J.*, vol. 21, no. 22, pp. 24910–24918, Nov. 2021.

[26] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Oct. 2018.

[27] "Understand Linux Load Averages and Monitor Performance of Linux." 2017. [Online]. Available: https://www.tecmint.com/understand-linux-load-averages-and-monitor-performance/

[28] B. Martinez, M. Montón, I. Vilajosana, and J. D. Prades, "The power of models: Modeling power consumption for IoT devices," *IEEE Sensors J.*, vol. 15, no. 10, pp. 5777–5789, Oct. 2015.

[29] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10700–10714, Dec. 2019.

[30] Q. Gao, K. J. Blow, D. J. Holding, I. W. Marshall, and X. H. Peng, "Radio range adjustment for energy efficient wireless sensor networks," *Ad Hoc Netw.*, vol. 4, no. 1, pp. 75–82, 2006.

[31] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, "Reliable federated learning for mobile networks," *IEEE Wireless Commun.*, vol. 27, no. 2, pp. 72–80, Apr. 2020.

[32] T. Meng, K. Wolter, H. Wu, and Q. Wang, "A secure and cost-efficient offloading policy for mobile cloud computing against timing attacks," *Pervasive Mobile Comput.*, vol. 45, pp. 4–18, Apr. 2018.

[33] T. Meng, H. Wu, Z. Shang, Y. Zhao, and C.-Z. Xu, "CoOMO: Cost-efficient computation outsourcing with multi-site offloading for mobile-edge services," in *Proc. 16th Int. Conf. Mobility Sens. Netw. (MSN)*, Tokyo, Japan, 2020, pp. 113–120.

[34] "Encrypted Training with Pytorch + Pysyft." 2019. [Online]. Available: https://blog.openmined.org/encrypted-training-on-mnist/

[35] Q. Zhu, B. Si, F. Yang, and Y. Ma, "Task offloading decision in fog computing system," *China Commun.*, vol. 14, no. 11, pp. 59–68, Nov. 2017.

[36] A. Bozorgchenani, D. Tarchi, and G. E. Corazza, "An energy and delay-efficient partial offloading technique for fog computing architectures," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, 2017, pp. 1–6.

[37] "Intelligent Visual Inspection, Anywhere: IBM Power Systems." [Online]. Available: https://www.ibm.com/downloads/cas/OGPLXPWJ (Accessed: Jul. 7, 2021).