Pervasive and Mobile Computing 76 (2021) 101459

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

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

REAM: A Framework for Resource Efficient Adaptive Monitoring of Community Spaces

Praveen Venkateswaran [a,*], Kyle E. Benson [b], Chia-Ying Hsieh [c], Cheng-Hsin Hsu [c], Sharad Mehrotra [a], Nalini Venkatasubramanian [a]

[a] University of California, Irvine, USA
[b] Real-Time Innovations, Sunnyvale, USA
[c] National Tsing-Hua University, Taiwan

ARTICLE INFO

Article history:
Received 12 January 2021
Received in revised form 5 June 2021
Accepted 5 August 2021
Available online 13 August 2021

Keywords:
Smart Community
IoT
Edge computing
Adaptive Monitoring
Reinforcement learning

ABSTRACT

Nowadays, many Internet-of-Things (IoT) devices with rich sensors and actuators are being deployed to monitor community spaces. The data generated by these devices are analyzed and turned into actionable information by analytics operators. In this article, we present a *Resource Efficient Adaptive Monitoring* (REAM) framework at the edge that adaptively selects workflows of devices and analytics to maintain an adequate quality of information for the applications at hand while judiciously consuming the limited resources available on edge servers. Since community spaces are complex and in a state of continuous flux, developing a one-size-fits-all model that works for all spaces is infeasible. The REAM framework utilizes reinforcement learning agents that learn by interacting with each community space and make decisions based on the state of the environment in each space and other contextual information. We demonstrate the resource-efficient monitoring capabilities of REAM on two real-world testbeds in Orange County, USA and NTHU, Taiwan, where we show that community spaces using REAM can achieve $>$ 90% monitoring accuracy while incurring $\sim$ 50% less resource consumption costs compared to existing static monitoring approaches. We also show REAM's awareness of network link quality in its decision-making, resulting in a 42% improvement in accuracy over network agnostic approaches.

1. Introduction

Modern community spaces, such as classrooms, buildings, campuses, districts, and cities, are increasingly instrumented with Internet-of-Things (IoT) sensors and actuators, which enable many real-time community monitoring applications. Monitoring applications are now ubiquitous across many domains like smart transportation, smart water infrastructure, environmental sensing, among others [1,2]. A typical monitoring workflow involves collecting data through sensors deployed in the community spaces and analyzing this data to generate meaningful and actionable information. While the collected IoT sensor data can be sent to cloud data centers for analysis, oftentimes this can lead to high operational costs, slow response times, and service interruptions, as the data centers are often far away from the community spaces [3]. An alternate solution is to leverage *edge servers* which can be network gateways or dedicated workstations, that are in closer proximity [4] for less expensive, more responsive, and more reliable analysis results. In addition, many community

* Corresponding author.
E-mail address: praveenv@uci.edu (P. Venkateswaran).

https://doi.org/10.1016/j.pmcj.2021.101459
1574-1192/

agencies have privacy and security policies that prevent them from using a public cloud provider, and instead require solutions that can be situated on servers located on-premise. The deployment of an edge driven solution is important for monitoring applications run by these public works agencies.

In this article, we refer to the algorithms that analyze the sensor data into analysis results as *analytics*, which are composed of multiple *operators* that can be dynamically deployed on different networked edge servers. These operators can range from simple rule-based heuristics to complex machine learning models. System administrators or community stakeholders could concurrently execute multiple monitoring applications using different sensors and analytics at any time. However, the quality of network links over which the sensor data and analytics results are sent can impact the effectiveness of the monitoring applications due to the loss of data packets, low bandwidth availability, etc. Given that the applications, sensors, analytics, networks, and edge servers are all highly heterogeneous, effective sensor/analytics selection and efficient resource allocation are key to the success of smart community monitoring applications.

In this article, we propose a *Resource Efficient Adaptive Monitoring (REAM)* framework to dynamically select the sensors/analytics to execute at the edge, in order to meet the objectives of multiple monitoring applications in a community space. Realizing REAM is no easy task due to the following challenges:

- **High interoperability.** Most community spaces are progressively deployed with high heterogeneity. REAM has to ensure efficient data and information exchange between IoT devices and edge servers.
- **High flexibility.** The objective of each monitoring application can be met with different workflows of sensors and analytics, each of which would require a certain amount of compute, networking, and other resources, and provide a certain quality of results [5,6]. REAM therefore should leverage the different options of sensors and analytics, while meeting the quality requirements of each monitoring application.
- **High adaptability.** Community spaces are complex and dynamic, while events in different spaces can take place under different contexts due to differences in location, demographics, structure, etc. It is, therefore, extremely challenging to develop accurate rules or models for each individual space. It is also important for the framework to be able to make online decisions with noisy inputs and to work well under diverse network conditions and resource availability. Making adaptation decisions, therefore, is a key objective of our REAM framework.

Our earlier work [7] presented an initial framework design for REAM. In this article, we have taken several steps toward a real-world deployment of the REAM framework to address the above challenges. To achieve *interoperability*, we adopt the publish–subscribe data exchange model to facilitate efficient information exchange. This provides a robust methodology for control and data flows within the REAM architecture, where devices, agents, and edge nodes can communicate with one another allowing for users to easily deploy and manage REAM in a distributed environment. We also partnered with Real Time Innovations [8] and use their software to implement data exchange in our REAM deployments. Secondly, in addition to sensing and analytics, we also incorporate network quality awareness in REAM's decision making for better *flexibility*. Most real-world deployments have heterogeneous modes of connectivity such as WiFi, Bluetooth, etc., which offer varying levels of Quality-of-Service (QoS). Moreover, the quality of connectivity is not constant and can vary based on the network utilization and data transmission of other devices and applications. Incorporating network quality awareness is an important step toward a practical and more realistic approach to real-world deployments which we demonstrate through experimental evaluations in this article. To the best of our knowledge, most prior efforts to monitor community spaces with IoT analytics assume each application only comes with a predetermined workflow of sensors/analytics [2,9]. For high *adaptability*, we apply Reinforcement Learning (RL) [10] to develop a *decision making algorithm* for the most appropriate adaptation decisions at any given moment. Our RL-based algorithm employs *agents* to directly learn from experience by interacting with the diverse and dynamic environments. We demonstrate how our design choices enable seamless integration of the RL agents with the publish–subscribe data exchange, as well as how the incorporation of network quality awareness is critical for improved decision making in real-world community spaces. We extensively evaluate REAM on real world testbeds and demonstrate its effectiveness across diverse applications and deployments, in addition to evaluating its scalability for large-scale deployments.

This article makes the following contributions.

- We design a novel REAM framework that jointly considers IoT sensors, analytics operators, and network links at the edge when monitoring community spaces.
- We formulate a decision making problem for the REAM framework to make adaptation decisions under the constraints of resource availability and network quality. We then present our RL-based algorithm and show case two real monitoring applications: stormwater contamination monitoring and pedestrian counting, while many other monitoring applications are possible.
- We evaluate our REAM framework on two real-world testbeds in Orange County, USA and NTHU, Taiwan and compare it to baseline approaches, when assuming the network loss is negligible. We show that REAM can achieve $>$90% monitoring accuracy while incurring $\sim$50% lower resource consumption costs compared to existing static monitoring approaches.
- We also conduct detailed simulations to validate the network-quality awareness of our framework under different network loss rates as well as its scalability to larger-scale deployments. We demonstrate that REAM can achieve up to 42% improvement in accuracy over static approaches by being cognizant of and adapting to differing network quality conditions. We also show that REAM can provide near real-time service for a multitude of monitoring applications, and incurs only a minimal increase in runtime ($<$0.002 s), for even large scale deployments with over 1000 nodes.

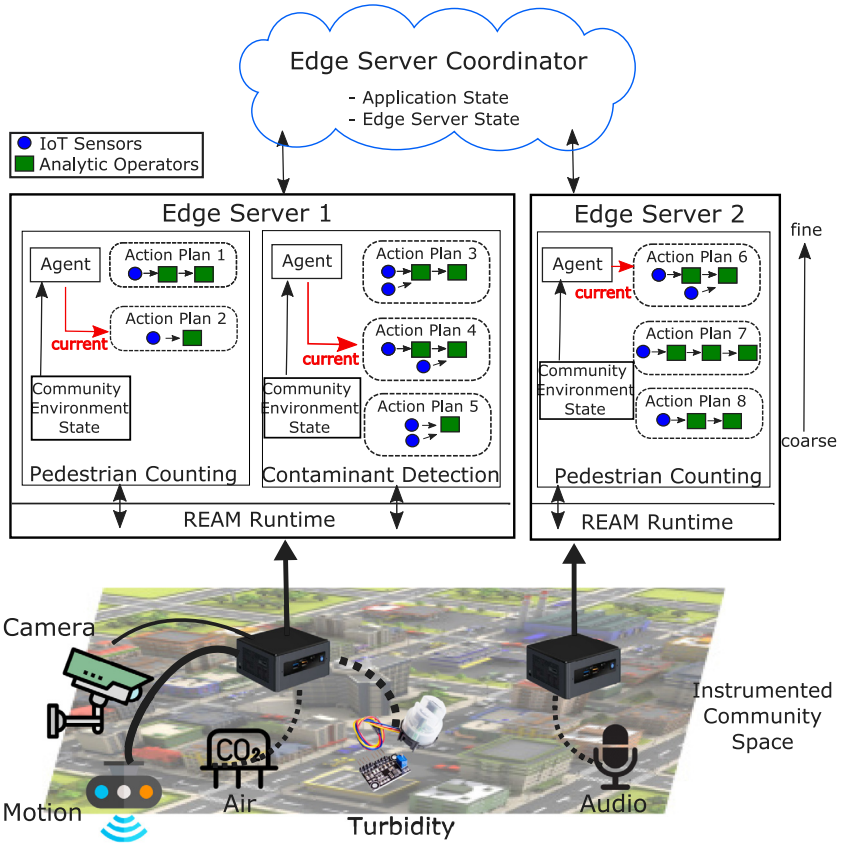

Fig. 1. REAM monitoring framework architecture.

Section 2 describes the architecture and workflow of REAM, and we present our formulation and detailed approach in Section 3. We describe our real-world testbeds and present our experimental results in Section 4. Section 5 discusses existing related work. We present our conclusion and future work ideas in Section 6.

2. REAM overview

In this section, we describe the design choices and architecture of our proposed REAM framework. In order to optimize the selection of sensors, analytics operators, and their associated network links, we organize them into workflows. The choice of using RL agents over a supervised learning approach is due to the fact that community behavior and patterns often change over time and hence the definition of events in a space can change. It is also not always obvious what the right sensing and analytics option is and may require a sequence of selections, and hence RL provides more flexibility in the way an agent can be trained to adapt to changing conditions. In order to support the publish–subscribe data exchange described earlier, we develop a middleware at each edge server and also define a coordinator between edge servers.

2.1. REAM architecture

Fig. 1 illustrates our proposed monitoring framework with two edge servers that have three sample monitoring applications running on them. Each monitoring application relies on the measurements of a specific set of sensors that have been instrumented in the community space. The communication and data transmission between the sensors and the edge server can take place through various networks like WiFi, Bluetooth, ZigBee, LoRa, Ethernet, optical fiber, etc. These network links can have differing quality levels captured by network loss rate, available bandwidth, etc, depicted by the dashed lines in Fig. 1. Once the sensor data are received at the edge server, they are run through a set of analytic operators which could constitute ETL (Extract, Transforming, Load) functions, Machine Learning models, Time-series analysis, among others in order to obtain useful information. Next, we introduce several key software components in REAM architecture.

Action plans. Since each application can use different combinations of sensors and analytics to achieve its objective with differing quality of results, we define each combination as an *action plan* where each plan can be thought of as a *workflow*,

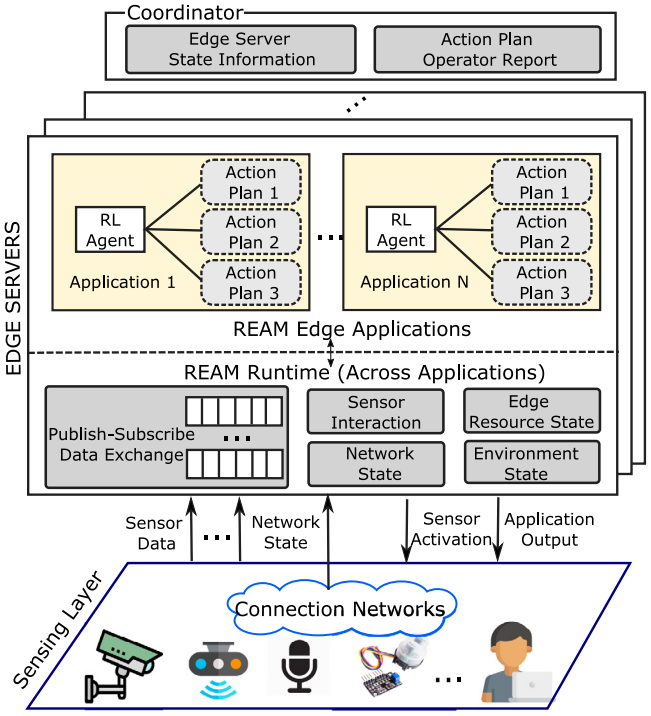

Fig. 2. REAM uses a publish–subscribe data exchange model for control and data flows.

or an execution graph, of sensors, analytic operators, and network links. They can vary in their execution complexity (large workflows with numerous sensor inputs and analytics), their resource requirements (resource-heavy sensors, large data volumes, complex analytics models), as well as in the quality of their network links (links with high loss rates would result in more packets being dropped). In our framework, as illustrated in Fig. 1, every monitoring application is a collection of action plans which can be a *coarse-grained* action plan that provides a baseline quality of continuous monitoring while consuming less resources, or various *fine-grained* action plans that provide a range of in-depth monitoring at higher costs, providing better results.

RL agents. In the REAM framework, at each timestep, an application can choose to execute one of its action plans as denoted by the *current* tags in Fig. 1. The decision of which action plan to execute is taken by an RL agent that learns by interacting with the community space based on its application's objective. The agent observes the readings from the application's sensors, network link quality information, outputs of the analytic operators, and other external community space contextual information such as the weather and time-of-day. It uses this information to develop a probabilistic learning model that drives the selection of which action plan to execute based on the effectiveness or utility of each plan.

Our framework design assigns one agent for each application. We opt not to create a global agent across all applications at the edge for the following reasons - (1) *Flexibility:* Individual agents simplify the process of dynamically adding or removing monitoring applications since agents can be trained independently of others unlike with a global agent; (2) *Tractability:* The dynamic and complex nature of community spaces can result in the agent having to reason about an extremely large number of states [11]. By assigning one agent to each application, we can ensure that the number of states is manageable; (3) *Simplicity:* Applications can have different objectives and operate at different time granularities. It is therefore difficult to define a global objective for each community space that is normalized across different applications. Furthermore, individual agents allow each application to set its own timestep granularity for sensing, monitoring, and analysis, despite the resulting decisions may slightly differ from the optimal ones. Hence, we opt for individual agents.

REAM runtime. In order to facilitate the exchange of information — both data and control messages, we implement a middleware on each edge server called REAM Runtime. As shown in Fig. 2, the REAM Runtime hosts the publish–subscribe data exchange implementation of REAM, and handles the information flow of data from the community environment to the edge server, capturing the network and environment states. In addition, the REAM Runtime middleware also tracks the amount of resources available on the edge server which is an essential information for the RL agent to be able to decide on feasible action plans that it can utilize. Finally, REAM Runtime also communicates back to the devices and

Table 1
Symbols used in the article.

Symbol	Description
$\mathcal{A}$	Set of monitoring applications
$\mathcal{S}$	Set of sensors
a_i^{ϕ}	Priority of application a_i
$\mathcal{O}$	Set of analytic operators
$\mathcal{L}$	Set of network links
$\mathcal{P}$	Set of action plans
$\mathcal{B}(p_j)$	Benefit of action plan p_j
$\mathcal{C}(p_j)$	Cost of action plan p_j
R_k	Amount of resource of type k
$\mathcal{N}(p_j)$	Network loss on links of action plan p_j
$\mathcal{U}(p_j)$	Utility of an action plan p_j
r_t	Reward at time step t
$\mathcal{S}'$	Operating state of the sensor servicing the application
$\mathcal{O}'$	Analytic operators currently running
$v(\mathcal{A}')$	Value returned by the analytic operators
$\mathcal{N}'$	Loss rates of the network links used by the application
Ext	External contextual information about the community space
$\mathcal{P}'$	Set of valid action plans
J_t	Cumulative reward

analytics operators that have been selected by the RL agent using the publish–subscribe paradigm. We further detail the control and data flows in REAM in the next section.

Edge server coordinator. In order to maintain a repository of the available action plans and the state of each edge server and its applications, we design an Edge Server Coordinator that can reside in the cloud or on an edge server. It also maintains knowledge of (1) the application states including their action plans, resource requirements, and objectives, and (2) the edge server states which include their resource availability, current applications, action plans, and corresponding network link quality. Agents can use such information to take decisions, and system administrators can use the coordinator to modify application objectives and resource availability.

2.2. REAM control/data flows

Fig. 2 presents the control and data flows in our REAM framework. In order to facilitate information and data exchange among the components, the REAM framework architecture uses a *publish–subscribe* data exchange model. This model allows sensors, edge servers, network links, analytic operators, and the coordinator to have their own topics (or channels) that they publish information to, which can be accessed by subscribing to these topics. In particular, sensors publish their raw data streams and network link quality information to dedicated topics, which are subscribed to by edge servers hosting one or more analytic operators that analyze the sensor's data. The computed analytics results by these operators are then published to dedicated topics for each application, that can be subscribed to by community stakeholders and administrators. The action plans selected by the RL agents are also published in order to activate the appropriate sensors and operators in the selected plan by the REAM Runtime. It also communicates with the coordinator to exchange resource availability, action plan selection, and metadata information.

3. REAM formulation

In this section, we formulate the problem of resource efficient adaptive monitoring in community spaces, describe our approach to represent the decision making as an RL task, and then present our solution approach. We provide a list of symbols used throughout this article in Table 1.

3.1. Utility driven action plan selection

We consider a community space that has a set of monitoring applications $\mathcal{A}$, where each application $a_i \in \mathcal{A}$ has a priority a_i^{ϕ} associated with it. For example, a gunshot detection application in a community would have a higher priority than a parking violation monitoring application. The community space is instrumented with a set of sensors $\mathcal{S}$, whose data can be analyzed using a set of analytic operators $\mathcal{O}$ that are transmitted over a set of network links $\mathcal{L}$ and are hosted on a set of edge servers.

We define a set of *action plans* $\mathcal{P}$, where each plan $p_j \in \mathcal{P}$ consists of a workflow of sensors, network links, and analytic operators, and services a specific application. Each action plan p_j provides a certain *benefit*, $\mathcal{B}(p_j)$, for the monitoring application it services which is dependent on the application's objective. Each plan p_j also incurs a *cost* $\mathcal{C}(p_j)$ which reflects the amount of resources R_k of type k (e.g., CPU, bandwidth, power, memory, etc.), that it consumes to run all the sensing

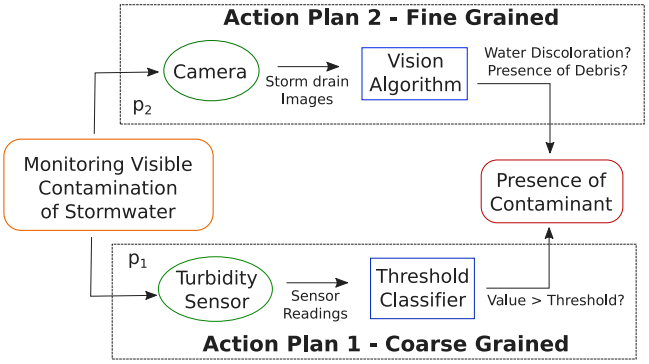

Fig. 3. Example of two action plans for a stormwater visible contamination monitoring application.

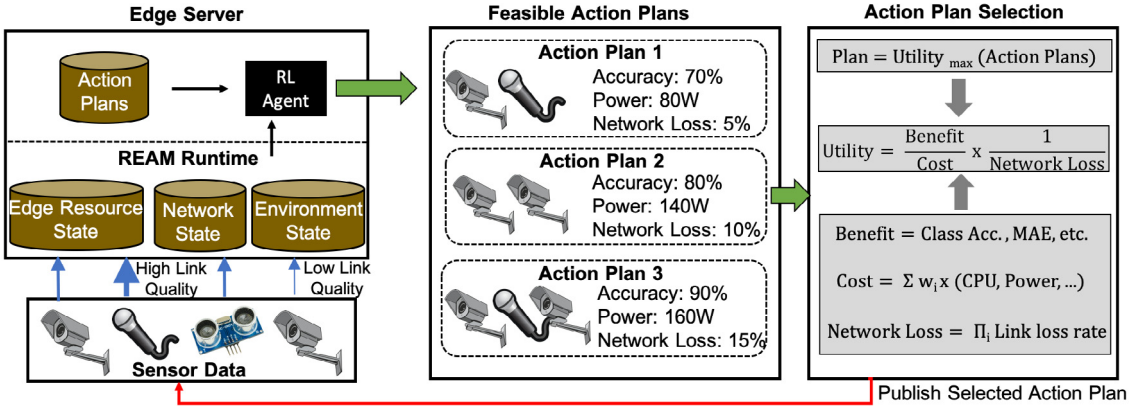

Fig. 4. REAM action plan selection workflow.

and analytics present in the action plan. Each plan also incurs a *network loss* $\mathcal{N}(p_j)$ reflecting the network loss rates across the links $l_i \in \mathcal{L}$, connecting the devices in the plan. We then define the overall utility of an *action plan* p_j as:

$$\mathcal{U}(p_j) = \frac{\mathcal{B}(p_j)}{\mathcal{C}(p_j)} \times \frac{1}{\mathcal{N}(p_j)}. \tag{1}$$

Fig. 3 shows an example of two different action plans that service the same stormwater visible contamination monitoring application. Plan p_1 utilizes a simple turbidity sensor that would be less accurate than the camera based solution of plan p_2, since it relies on a manually set and potentially erroneous threshold. Moreover, today's state-of-the-art vision algorithms can typically achieve high levels of accuracy and thus p_2 can provide a much higher benefit to the application. However, the cost incurred by p_1 is much lower than that of p_2, since the periodic capture and transmission of images would consume a lot of network bandwidth, the camera would require more power, and the vision algorithm would also consume more compute resources in order to provide results in near real-time.

Moreover, the *benefit vs. cost* tradeoff captured by the utility of an action plan, is also dependent on various environmental contexts of the community space which REAM leverages on. For instance, at night, the camera images may not be good enough for the vision algorithm to detect discoloration and debris, which might result in both action plans having similar accuracy. Hence, it would be a prudent decision to execute the coarse-grained plan more frequently at night since it can achieve similar benefit at lower costs, and the fine-grained plan during the day when it can provide much higher benefit. Furthermore, it is also important to consider the quality of the network links across which data is transmitted by each plan. If the links in plan p_2 result in a large number of packets being dropped, this can cause a drop in monitoring accuracy due to the lack of data delivery, even though the camera images provide high quality information, and hence using the turbidity sensor based plan p_1 might provide higher utility due to consistent data delivery.

We visualize the action plan selection workflow used by REAM in Fig. 4. Each edge server receives sensor data and network link quality information through the publish–subscribe data exchange model. The RL agent in the edge server then leverages this information, along with knowledge of the environment states, current resource availability of the edge server, as well as available action plans, in order to generate a set of action plan candidates along with their expected benefit and cost attributes. It then computes the relative utility of each plan and selects the plan with the highest

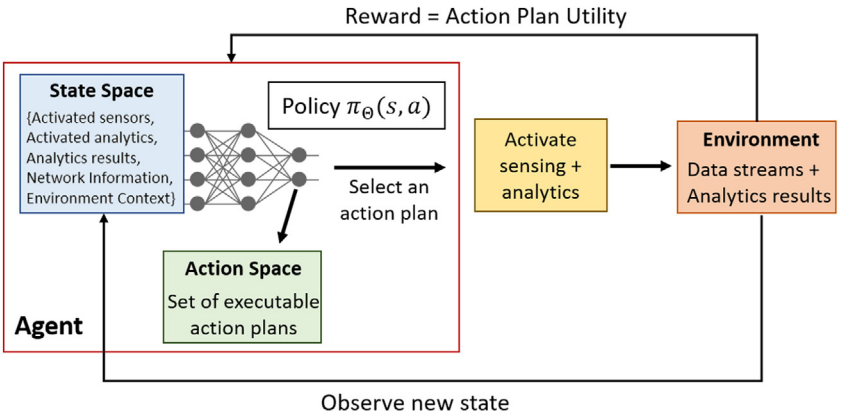

Fig. 5. REAM RL with DNN driven policy.

utility. Once the plan has been selected by the agent, the REAM Runtime middleware then publishes this information to community stakeholders as well as to activate the sensors in the selected action plan.

3.2. Defining RL agents in REAM

Consider the general setting shown in Fig. 5, where an RL *agent* interacts with an *environment*. At each time step t, the agent observes some state s_t, and then chooses to perform an action a_t based on a policy. Once the action is performed, the environment transitions its state to s_{t+1} and the agent receives a reward r_t. The state transitions and rewards are stochastic and are assumed to have the Markov property, i.e., the state transition probabilities and rewards depend only on the state of the environment s_t and the action a_t taken by the agent.

State space. In the REAM framework, we represent the state s_t of each application's RL agent at any given time as a class object that consists of the following attributes - (1) $\mathcal{S}'$: the operating state of the sensors servicing the application, (2) $\mathcal{O}'$: the analytic operators currently running, (3) $v(\mathcal{A}')$: the value returned by the analytic operators, (4) $\mathcal{N}'$: the network loss rates of the network links used by the application, and (5) *Ext* : external state and contextual information about the community space (e.g., time-of-day, weather information, etc.), which can influence the performance of the sensors and analytic operators.

Action space. At each timestep, an application's agent determines its action space as a set of valid action plans $\mathcal{P}' \subseteq \mathcal{P}$ that it could potentially execute from its current state. The timestep is configurable, which can be different for individual applications in the space. Each plan $p_j \in \mathcal{P}'$ consists of a set of active sensors, their operational states (on/off for simple sensors, pan–tilt–zoom for a camera), the network links used by devices in the plan, and a set of active analytic operators together with its workflow.

Reward. The reward r_t obtained by the agent for executing an action plan is the utility provided by that plan. The benefit of plan p_j depends on the specific application (e.g., classification accuracy, distance based error, etc.). We compute the cost of p_j by first normalizing the amount of resources required of each resource type (CPU, bandwidth, memory, etc.) across all action plans of the application and then calculating a weighted sum of these normalized costs for plan p_j as: $\mathcal{C}(p_j) = \sum_{k=1}^{|R|} w_k \times R_k^{p_j}$, where w_k refers to the weight and $R_k^{p_j}$ refers to the normalized amount of resources of type k required for plan p_j. The weights allow system administrators to prioritize the conservation of certain types of resources and lessen their importance if they are abundantly available. We then compute the network loss for p_j as the multiplication of the loss rates across all the links, $l_i \in \mathcal{L}$ used by the plan, i.e., $\mathcal{N}(p_j) = \prod_{i=1}^{n} l_i$.

3.3. Training RL agents

Each agent can only control its action plan selection and has no apriori knowledge of the rewards or the state transitions which can be affected by external factors. During training, the agent interacts with the community space environment and observes the rewards and state transitions while choosing different action plans. The agent's goal is to select action plans in a way that maximizes the cumulative reward J_t it receives over any time period T, i.e., $J_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where γ is a discount factor $\in [0, 1]$ and $r_{t'}$ is the reward at timestep t'. We then define $Q^*(s, p_j)$ as the maximum expected reward achievable by following a policy $\pi(s, p_j)$, which refers to the probability of action plan p_j being chosen by the agent when in state s. That is, $Q^*(s, p_j) = \max_\pi \mathbb{E}[J_t | s_t = s, p_{j_t} = p_j, \pi]$. Using the Bellman equation [12], this can be represented as:

$$Q^*(s, p_j) = \mathbb{E}[r_t + \gamma \max_{p_{j_{t+1}}} Q^*(s_{t+1}, p_{j_{t+1}}) | s_t, p_{j_t}].$$

Since community spaces are complex and can have a large number of possible {state, action plan} pairs, it would be infeasible to store the policy in a tabular form as a lookup table, also known as a Q-table. Instead, it is more common to use *function approximators* to represent the policy by estimating $Q^*(s, p_j)$. Among the approximators, Deep Neural Networks (DNNs) [13] have recently gained popularity for solving large-scale RL tasks since they do not need hand-crafted weights. A network with weights θ can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ at each iteration i, where:

$$L_i(\theta_i) = \mathbb{E}\left[\left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1}) - Q(s_t, a_t; \theta_i)\right)^2\right].$$

Algorithm 1 Deep Q-learning Algorithm

1: Initialize Replay Buffer $\mathcal{D}$
2: Initialize Q, DNN with random weights θ
3: **for** $t = 1 \rightarrow T$ **do**
4: With probability ϵ, select a random *action plan* p_j otherwise, select $p_j = \max_p Q(s_t, p; \theta)$
5: Communicate chosen plan with REAM Runtime and receive allowed plan p_j'
6: Execute *action plan* p_j' and observe environment to get reward r_t and state s_{t+1}
7: Store transition $(s_t, p_j', r_t, s_{t+1})$ in $\mathcal{D}$
8: Sample random minibatch of transitions (s_k, p_k, r_k, s_{k+1}) from $\mathcal{D}$
9: Set $y_j = r_j + \gamma \max_p Q(s_{t+1}, p; \theta)$
10: Perform gradient descent step on $(y_j - Q(s_t, p_j; \theta))^2$ with respect to θ
11: **end for**

We represent the action plan decision making policy as a neural network with weights θ which takes the current state of the RL agent as input and outputs a probability distribution over all valid potential next action plans. Note that we allow the RL agent to continue executing the current action plan in the next timestep as well.

We train the agents using the deep Q-learning algorithm [14] as shown in Algorithm 1. It uses an ϵ-greedy policy [10] in order to select an action plan by either randomly selecting a plan p_j with a probability ϵ, or selecting the plan with the maximum value of the probability distribution. At each timestep, the chosen action plan is executed and its reward and the next state are observed. We store the agent's transitions in a buffer $\mathcal{D}$ of a fixed size and then perform gradient descent to update the weights θ of the neural network (Line 10) using a minibatch of transitions drawn at random from the buffer as

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{i-1}) - Q(s_t, a_t; \theta)\right)\nabla_{\theta_i} Q(s_t, a_t; \theta_i)\right].$$

Rather than computing the full expectations in the above gradient, it is often computationally expedient to optimize the loss function using approaches like stochastic gradient descent, Adam [15], RMSProp [16], etc.

3.4. Prioritized action plan selection

Before the RL agents identify the optimal action plans, the edge server must perform a sanity check to find the subset of all action plans that can be feasibly executed without saturating all the available resources. That is, the REAM Runtime middleware at each edge server employs our proposed Prioritized Action Plan Section (PAPS) algorithm, as shown in Algorithm 2, to coordinate with the RL Agents in selecting action plans for varying resource availabilities and publishing that information to activate the appropriate sensors and analytic operators in the selected plans.

More specifically, at each timestep, the REAM Runtime sequentially communicates with each application's RL agent in the order of their priority $\mathcal{A}^{\phi}$. It limits the candidate action plans for each agent ($\mathcal{P}' \in \mathcal{P}$) based on the current resource availability. The agent then selects an action plan with the highest utility (Eq. (1)), based on the current state of the community environment. The REAM Runtime then obtains the selected action plan p_j, and updates the available resources ($\mathcal{R}$) as well as publishes this information to activate the sensors and operators that are parts of the selected plan p_j. If an edge server has limited resource availability, the algorithm ensures that high priority applications can select action plans with high utility, while potentially limiting the plans selected by low priority applications.

At every timestep, Algorithm 2 must handle all monitoring applications in the environment ($|A|$), each of which can have $|P|$ number of action plans. This can result in the need to publish to $|E|$ number of sensors and operators where E represents the total set of sensors and operators in the environment. This results in a polynomial time complexity of $O(PAE)$.

To illustrate our RL-based approach and the selection of action plans by the REAM RL agents, we give an example in Fig. 4. We see that the edge server receives sensor data from heterogeneous devices across network links with varying

Algorithm 2 Prioritized Action Plan Selection (PAPS)

Input: Action Plans $\mathcal{P} = \{p_1, ..., p_n\}$, Plan resource requirements $\mathcal{P}^{\mathcal{R}} = \{p_1^R, ..., p_j^R\}$, Application priority $\mathcal{A}^{\prec} = \{a_1^{\phi}, ..., a_j^{\phi}\}$, Available edge server resources $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_k\}$
for $t = 1 \rightarrow T$ **do**
  for $\phi = 1 \rightarrow j$ **do**
    Pass action plan candidates $\{\mathcal{P}' \in \mathcal{P} \mid (\mathcal{P}')^R > \mathcal{R}\}$ to agent a_i^{ϕ}
    Obtain action plan $p_j \in \mathcal{P}'$ with max. utility (Eq. (1))
    $\mathcal{R} = \mathcal{R} \setminus p_j^R$
    Publish to activate sensors and operators $\in p_j$
  end for
end for

quality. Algorithm 2 is used to determine the available resources on the edge server, and to determine the feasible action plans as shown in the third box. We can see that the three available action plans have varying benefits ($\mathcal{B}$), costs ($\mathcal{C}$) in terms of power consumption, and network loss ($\mathcal{N}$). To determine the reward of each plan, we normalize the values and use Eq. (1) to obtain a utility of [28.0, 9.1, 5.9] for the three action plans respectively. The RL agent hence chooses the first plan and publishes it. The sensors and analytics in the selected plan are then activated and this process repeats for the newly observed state from the sensors in the selected plan.

4. Experimental evaluation

In this section we evaluate the resource-aware monitoring capabilities of REAM and compare its performance to existing approaches. We present two monitoring applications in real-world testbeds located in Orange County, USA and NTHU, Taiwan.

In our first set of experiments (Sections 4.2 and 4.3) we evaluate the benefit vs. cost tradeoff in REAM. We assume that the network loss in these experiments are negligible. For each application, we compare the performance of REAM with static monitoring approaches that execute specific action plans in isolation. We also compare with a machine learning approach using random forest that uses the same training and test data.

We then evaluate how REAM performs in environments with network loss. Since a controlled introduction of loss in the real community testbeds is challenging, we simulate the input to the REAM prototype under conditions of network loss. We compare its performance with a network agnostic planning approach to evaluate the impact that network quality aware monitoring has on the monitoring accuracy (Section 4.4). We also evaluate the scalability of REAM to large-scale community deployments (Section 4.5)

4.1. Experimental setup

Prototype implementation. We implemented our REAM prototype shown in Fig. 2 using Python.[1] The RL agents are implemented using Keras [17], where each agent is a neural network containing two fully connected hidden layers with 24 neurons. We update the policy network parameters using the Adam algorithm [15] with a learning rate of 10^{-3} and implement our analytic operators for monitoring applications using Scikit-learn [18].

For the data exchange implementation in the prototype, although there are many publish–subscribe protocols available (e.g., AMQP, MQTT), the REAM prototype implementation uses the *Data Distribution Service* (DDS) protocol which offers several advantages. Unlike other protocols, DDS does not require centralized brokers and instead allows for fully decentralized, data-centric, and peer-to-peer communications. DDS supports dynamic discovery of publishers, subscribers, and data transmission while offering 30+ QoS levels for tuning data exchange performance, resource usage, priority, reliability, etc., which allows REAM to be fine-tuned for various environments. DDS also supports UDP for both high-performance and multicast communications. There are various DDS libraries available like OpenDDS, Opensplice, Fast DDS, etc. [19–21]. We chose to implement DDS in our prototype using the Real Time Innovations (RTI) Connext DDS framework [8] which provides specific implementations for resource-limited devices that are often found in smart community deployments.

Our prototype implementation defines publish–subscribe topics for each sensor and analytics output, coordinator messages, and every edge server using the `DDSTopic` interface found in the RTI Connext API [22]. Each topic is listened to using the `DDSTopicListener` interface, and also published to by implementing an object of `DDSPublisher` class associated with each sensor and analytics operator. The network monitoring through DDS is done using the

[1] https://github.com/praveenv/REAM.

Table 2
Power consumption of key devices.

Device	Make/Model	Power (W)	Note
Motion	Optex LX-402	0.33	
Camera	LiteOn 3MP	3	
PC	Intel i3 @ 1.7 GHz	6	Idle
PC	Intel i3 @ 1.7 GHz	27.5	Loaded
Stormwater	In-Situ 600	0.54	

Table 3
Resource consumption of video analytics.

Analytics	CPU usage	Memory usage	Running time per frame
OpenCV	194.80%	1.7%	0.052 s
YOLOv3	100.43%	8.5%	17.38 s

Fig. 6. Our testbed in Orange County, USA: a storm drain and the locations of sensing units.

`DDSFlowController` interface that also allows packets to be sent using a fixed rate or even on demand. Each edge server has an instance of such a network monitoring component implemented in the REAM Runtime middleware.

Resource measurements. Since the goal of the REAM framework is to be able to achieve application objectives while utilizing as little resources as possible, we capture the actual resource consumption (CPU, networking, and power) of the various devices and analytic operators in order to run faithful experiments when comparing our solution against baseline approaches. Tables 2 and 3 summarize the resource consumption of individual devices and analytic operators in both testbeds.

4.2. Smart stormwater infrastructure

We utilize five stormwater *sensing units* that have been instrumented by Orange County Public Works Department (OCPWD) in order to monitor the quality of the water flowing through the storm drains, which is depicted in Fig. 6. The stormwater can get contaminated while flowing into the drains by collecting pollutants like bacteria from human or animal waste, fertilizers, and even chemicals from industries that illicitly discharge their waste into these drains [23]. Each sensing unit consists of several hydraulic and chemical sensors to measure pH, turbidity, dissolved oxygen, flow rate, etc., that together are capable of detecting a wide range of potential contaminants. The sensor measurements are transmitted using LoRa networks and are analyzed at an edge server using Machine Learning classifiers to determine the presence of contamination. The sensing units are deployed in secure underwater housing and are battery powered. Accessing these units in order to replace the batteries, therefore, involves significant efforts to dig up the housing and access the hardware

Table 4
Stormwater contamination monitoring — Location A.

Comparison approach	Accuracy (%)	Total energy Consum. (J)	Exp. Batt. life (days)	Avg. detection delay (min)
REAM	90.9	86.40	53	20.2
Random forest	88.2	101.77	44	24.7
Fine-grained	95.4	155.52	29	14.4
Coarse-grained	73.3	46.08	98	45.9

Table 5
Stormwater contamination monitoring — Location B.

Comparison approach	Accuracy (%)	Total energy Consum. (J)	Exp. Batt. life (days)	Avg. detection delay (min)
REAM	80.1	95.76	46	14.5
Random forest	76.9	120.93	35	19.3
Fine-grained	86.1	163.44	27	10.1
Coarse-grained	68.6	54.21	90	24.6

within, hence frequent battery replacement would incur large costs. OCPWD's objective is to prolong the battery life while maintaining contamination event detection accuracy.

Since stormwater contamination events occur sporadically with long periods of normal activity, measurements of a subset of sensors can be sufficient to provide coarse-grained signatures that can then be used to trigger all the sensors for fine-grained monitoring during contamination events. This is because using all the sensors for continuous monitoring would consume a lot of battery power. The goal of deploying our REAM framework is to accurately identify stormwater contamination events while prolonging the battery life of these sensing units by appropriately switching between coarse and fine grained monitoring.

4.2.1. Stormwater data

We use four months of sensor measurements from the sensing units at two different locations (A, B). For each location, we use three months of data for training and one month for testing. The measurements have a granularity of 15 min and the contamination event ground truth was annotated by an expert from OCPWD. We also obtained precipitation data for the location and battery consumption information of the sensing unit. We use two sensors — dissolved oxygen and pH, that are most sensitive to changes in the ecosystem to form the *coarse-grained* baseline action plan along with a Support Vector Machine (SVM) classifier. But since the changes can be due to minor natural variances in the chemical composition of the water, the coarse-grained plan can result in a number of false-positives. We hence define one *fine-grained* action plan that uses more information, consisting of the previous sensors along with temperature, Total Dissolved Solids (TDS), conductivity, and turbidity sensors and uses a Random Forest classifier, that is triggered by the coarse-grained approach and can more accurately determine if a contamination event has occurred. We define the reward for the REAM RL agent based on the utility provided, where the benefit $\mathcal{B}(p_j)$ of every action plan is its classification accuracy and its cost $\mathcal{C}(p_j)$ is the total battery consumption of the sensors and analytic operators in the action plan.

4.2.2. Stormwater contamination monitoring results

We measured the accuracy achieved in classifying contamination events for the test data from both locations (Tables 4 and 5). We observed that REAM achieved 90.9% and 80.1% accuracy at location A and B respectively, which is comparable to the 95.4% and 86.1% achieved by using only the fine-grained action plan and better than the 88.2% and 76.9% obtained by using the Random Forest supervised learning approach and the 73.3% and 68.6% achieved by using just the coarse-grained action plan. Moreover, the REAM framework consumed on average 42% less energy than the fine-grained action plan across both locations, resulting in a longer battery life by 24 days at location A and 19 days at location B that we derived based on the two D-cell alkaline battery capacity of the In-Situ 600 stormwater sensing unit.

At location A, REAM had a 20 min delay on average in detecting contamination events, compared to the 14, 24 and 45 min delays achieved by the fine-grained, Random Forest, and coarse-grained approaches respectively. At location B, REAM's average detection delay was 14 min, compared to 10, 19 and 24 of the fine-grained, Random Forest and coarse-grained approaches respectively. Fig. 7 shows a zoomed in view of the contamination event ground truth and the action plans chosen by the REAM RL agent during a week of the test period. We observe that for most of the contamination events, the agent utilizes the fine-grained action plan to achieve high accuracy and ends up using the coarse-grained plan during periods when no events occur. The occasional shift to the fine grained plan as shown by the red circle occurs since the agent explores different action plans based on the ϵ-greedy policy described in Section 3.2 to adapt to changing environmental conditions, e.g., dry vs. wet weather, seasonal patterns, etc.

From these results, we can see that the REAM framework can increase the battery replacement cycle from less than 1 month with the fine-grained approach to almost 2 months with less than a 5% drop in accuracy and a detection delay within 5 min on average.

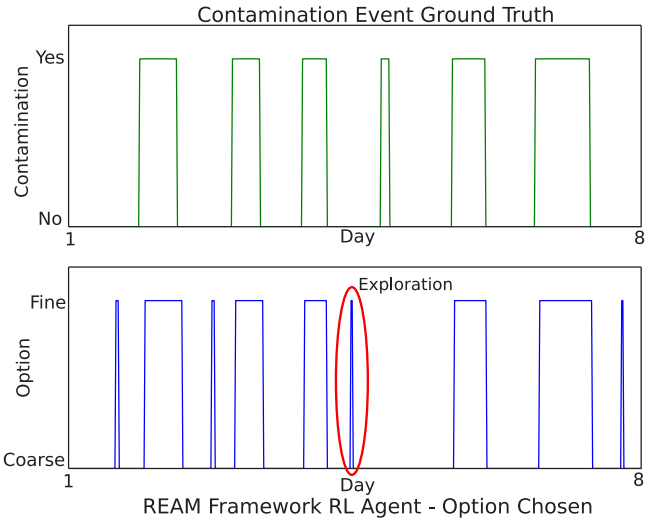

Fig. 7. The action plans selected by the REAL RL agent to determine contamination events.

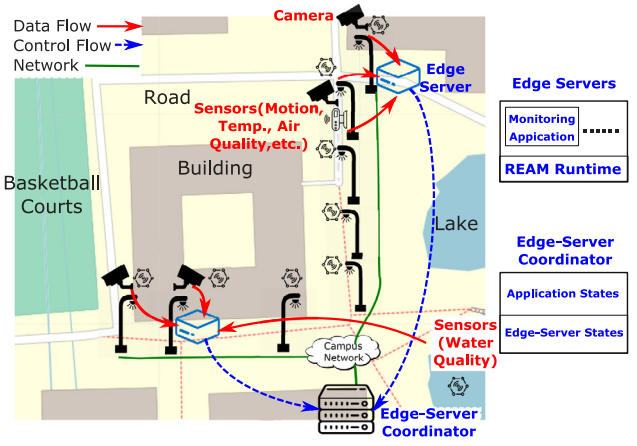

Fig. 8. Sensor-rich smart street lamps - NTHU campus, Taiwan.

Fig. 9. Our testbed at NTHU campus, Taiwan: a smart street lamp (left) and a motion sensor (right).

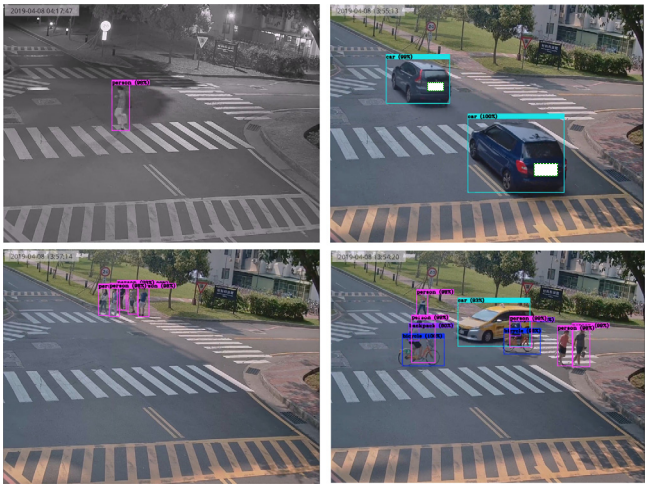

Fig. 10. Sample video frames from a street lamp camera for the pedestrian counting application. The recognized objects and bounding boxes are given by YOLOv3.

4.3. Smart campus

We have instrumented eight smart street lamps on the NTHU campus in Taiwan, as shown in Fig. 8 for smart campus applications. In our testbed, each street lamp is instrumented with a power supply, an Ethernet switch, a Raspberry Pi (which also serves as a Bluetooth and Zigbee gateway), and a wide spectrum of environmental sensors, such as motion (PIR, passive infrared), temperature/humidity, and air quality (PM 2.5) sensors. Four of the lamps are equipped with 3MP cameras, of which three are fixed bullet cameras and one is a Pan–Tilt–Zoom (PTZ) camera. There is also an outdoor motion sensor installed on one of the lamps, providing longer sensing range (12 m) for the intersection. Fig. 9 gives the photos of our NTHU testbed. The lamps are connected using a heterogeneous network consisting of Gigabit Ethernet, WiFi mesh, LoRa, and NB-IoT. We install edge servers in two of the street lamps for running monitoring applications. The edge servers are Intel NUC PCs, each has a 4-core CPU at 1.7 GHz, 8 GB RAM, and 500 GB disk.

We utilize this testbed for a pedestrian counting application that attempts to profile the movement of people at main intersections. This is to dynamically dispatch security guards to direct on-campus vehicles when intersections are crowded. The goal of the campus administration is to infer these profiles using as little resources as possible to ensure availability for other on-demand (emergency) applications. Using fine-grained camera feeds coupled with analytic libraries like YOLOv3 [24] and OpenCV [25] can result in accurate counts, but this approach is resource intensive. Since the flow of pedestrians is not continuous (fewer people walking at night), a coarse-grained motion sensor could be used to trigger the camera based analytics in order to conserve resources. However, since different moving objects (e.g., car, bicycle, etc.) can also activate the motion sensor, its accuracy would be lower than that of camera feeds. The goal of deploying our REAM framework is to be able to learn when pedestrians are likely to be present and switch between coarse- and fine-grained monitoring to preserve resources.

4.3.1. Pedestrian flow data

For the pedestrian counting application, we use two different sets of data collected over different time periods. The first dataset (termed Data1) consisting of video data from a camera and a motion sensor on a street lamp overlooking an intersection, was obtained over a week in April. The second dataset (termed Data2), utilized three week's worth of video and motion sensor data from the same street lamp during August. Using two such datasets also allows us to evaluate the effectiveness of REAM during extended periods of differing pedestrian flow. For Data1, we use five days for training and two days for testing, while for Data2 we use ten days for training and the remainder for testing. The measurements have a granularity of one second.

Fig. 10 shows four sample frames of the video data. For both datasets, we define the *coarse-grained* action plan to consist of the binary output analyzed from the motion sensor. This plan is sufficient to capture situations where there are none or just one pedestrian at a given time as shown in the top left frame of Fig. 10. However, we notice that the motion sensor can be triggered by other objects such as the vehicles in the top right frame resulting in false positives. Hence, we define two *fine-grained* action plans that run OpenCV [25] and YOLOv3 [24] object detection algorithms respectively, which can also handle cases where there are many pedestrians simultaneously present as shown in the bottom left frame. The YOLOv3 library is more powerful in that it can more accurately handle situations where there are multiple different objects like pedestrians and vehicles present together as shown in the bottom right frame. We hence assume that the output of the YOLOv3 plan is the ground truth for our evaluations. The benefit of every action plan is defined as its distance

Table 6
Pedestrian counting–Data1 (April).

Comparison approach	Distance from ground truth (%)	Total power consumption (W)	Total data generated (GB)
REAM	7.1	61.8	33.1
Random forest	15.4	54.6	30.3
YOLOv3	0	126.5	55.62
OpenCV	37.3	39.32	55.62
Motion sensor	62.3	36	0.0005

Table 7
Pedestrian counting–Data2 (August).

Comparison approach	Distance from ground truth (%)	Total power consumption (W)	Total data generated (GB)
REAM	8.6	319.9	160.5
Random forest	14.1	291.3	141.6
YOLOv3	0	697.5	305.9
OpenCV	32.1	246.2	305.9
Motion sensor	54.9	198	0.006

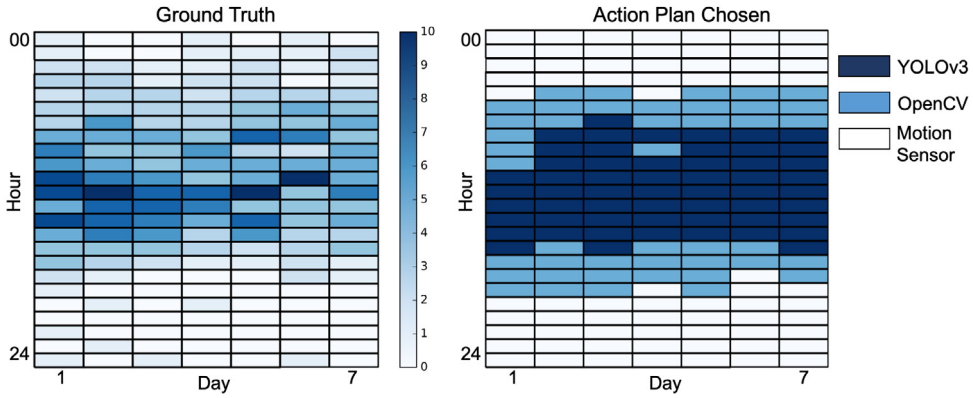

Fig. 11. Comparisons of hourly pedestrian count ground truth and action plan chosen by REAM framework during a week—Data1 (April).

from the ground truth in terms of the pedestrian count, and its cost is a weighted sum of its power, bandwidth, and CPU consumption. We assume equal weights in the evaluations if not otherwise specified.

4.3.2. Pedestrian counting results

Tables 6 and 7 show a summary of the performance comparisons, where we report the total power consumption as a sum of the power consumption of the sensors (motion, camera) and the edge servers. REAM achieved pedestrian count errors of 7.1% and 8.6% for Data1 and Data2, respectively, compared to the YOLOv3 based approach that we assumed to be the ground truth and performed better than the Random Forest, OpenCV, and the coarse-grained motion sensor based approaches for both Data1 and Data2. However, the YOLOv3 library is very resource intensive, and this coupled with the significant power consumption of using a camera continuously, results in REAM having 51.0% and 54.1% less power consumption over the two datasets. The REAM framework also results in 44.0% and 47.5% less data being generated than the YOLOv3 and OpenCV approaches that require continuous generation and transmission of video data.

Figs. 11 and 12 illustrate heatmap based comparisons of the ground truth of average hourly pedestrian flow per week and the corresponding most frequent action plan chosen by the REAM RL agent during that hour for Data1 and Data2, respectively. While we observe that the flow of pedestrians was slightly lower in Data2 (August) compared to Data1 (April), in both cases, the number of pedestrians is the highest during the day (8 am – 5 pm) and during those periods the predominantly used action plan is YOLOv3 which results in high accuracy. Also, during the night and early mornings, when extremely few pedestrians are on the road, the RL agent chooses to use the motion sensor approach which is sufficiently accurate to model the pedestrian flow. The REAM framework can thus achieve $>90\%$ accuracy (hence not missing many people), while consuming $\sim 50\%$ less power and generating less data (hence consuming less resources), compared to static monitoring approaches. The performance of the RL agent could be made better by including additional contextual information like the weather, campus holidays, etc., that would have a direct impact on the pedestrian flow.

4.4. Exploring network loss performance of REAM

We next evaluate the performance of our REAM prototype in the presence of network loss. Due to the challenge of experimenting with controlled network loss in a real community, we simulate the input and network loss to the

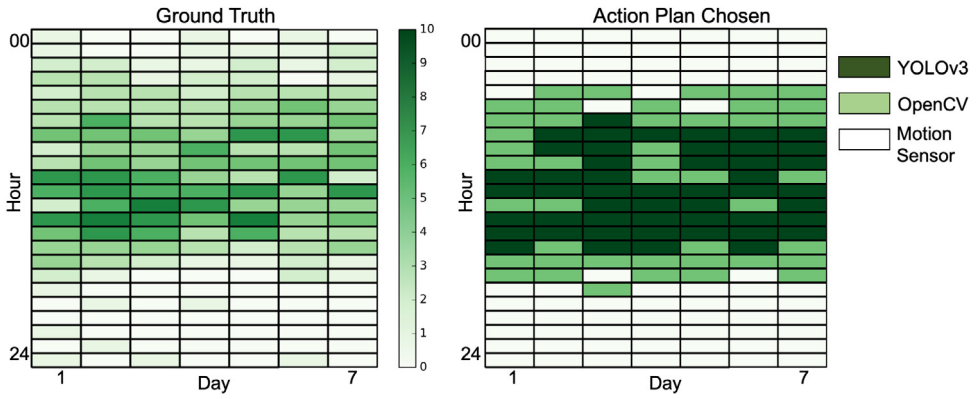

Fig. 12. Comparisons of hourly pedestrian count ground truth and action plan chosen by REAM framework during a week—Data2 (August).

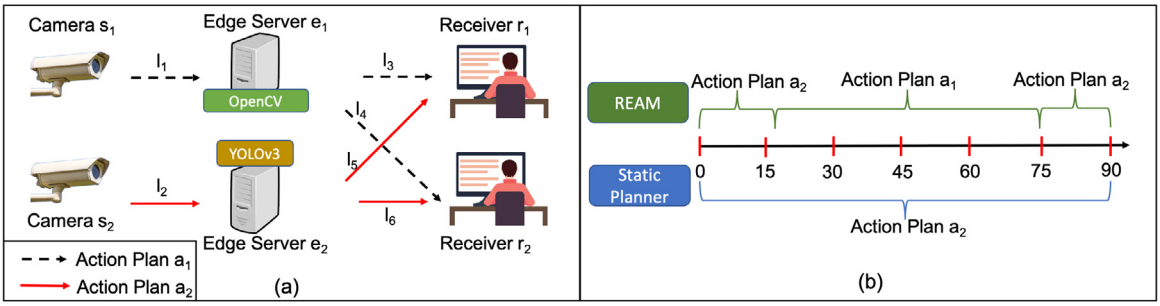

Fig. 13. Network quality simulator: (a) experiment setup and (b) action plans selected by the REAM and static planner.

prototype based on our testbed and compare the performance of REAM with a network agnostic planning approach. We use images from camera sensors with two different analytic operators YOLOv3 and OpenCV. As described earlier, while YOLOv3 achieves higher levels of accuracy, it uses more resources in terms of power consumption, CPU, and memory, as compared to OpenCV. In addition, we also leverage a feature in RTI's DDS framework to track the number of messages lost or received across all action plans over time.

In order to evaluate REAM under conditions with network loss, we define two action plans as shown in Fig. 13(a). The setup consists of 2 camera sensors $\{s_1, s_2\}$, 2 edge servers $\{e_1, e_2\}$, and 2 receivers $\{r_1, r_2\}$, with network links $\{l_1, \ldots, l_6\}$ connecting them as shown in the figure. There are 2 action plans $\{a_1, a_2\}$ that can be chosen, where a_1 uses OpenCV, and a_2 uses Yolov3 as the analytics operators respectively and hence benefit $\mathcal{B}(a_2) > \mathcal{B}(a_1)$. We assume that the sensors and edge servers are homogeneous which means that the cost $\mathcal{C}(a_2) = \mathcal{C}(a_1)$. When each experiment begins, the nodes start publishing data, and after 15 s, we introduce network loss on Link l_2. The loss rate persists for 60 s, after which it is set back to zero.

We compare the performance of REAM's network aware action plan selections with a static planning approach that uses Benefit and Cost to determine the action plan to use, but is agnostic to variability in network link quality, i.e., $\mathcal{U}_{\text{static}} = \mathcal{B}(p_j)/\mathcal{C}(p_j)$. Fig. 13(b) shows the action plan selections during the experiments by both approaches. We see that for the initial period with no network loss, both approaches choose action plan a_2 which has higher utility than a_1, since its benefit is higher and the cost and network loss is the same. However, when the network loss is introduced on Link l_2, REAM uses this information and selects a_1, whose utility becomes larger due to the network loss. However, the static planning approach continues to use a_2. When the network quality is restored, REAM once again switches back to action plan a_2 whose utility is once again larger than that of a_1.

For the above experiments, we first compare the cumulative data delivery rate over the affected Link l_2 achieved by REAM and the static planner for two different network loss rates of 0.1 and 0.5. As shown in Fig. 14, the loss incurred by REAM is significantly lower as compared to that by the static planner. The difference ranges from 6% for 0.1 network loss rate to 27.5% for 0.5 network loss rate. Since this difference in delivery rate is a function of the network loss rate on the affected link, the link quality of the alternative paths, as well as the duration for which the link disruption persists, it would be even bigger for large-scale disruptions on multiple network links.

We then compare the benefit achieved over time by REAM and the static planner during the experiments. We measure the accuracy over time by multiplying the expected accuracy of each selected action plan with the measured message delivery rate. We calculate the delivery rate for each action plan as the number of packets received each timestep divided

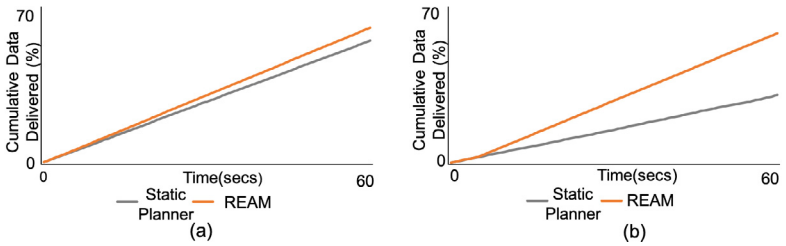

Fig. 14. Cumulative data delivered by the REAM and static planner at the network loss rates of: (a) 0.1 and (b) 0.5, respectively.

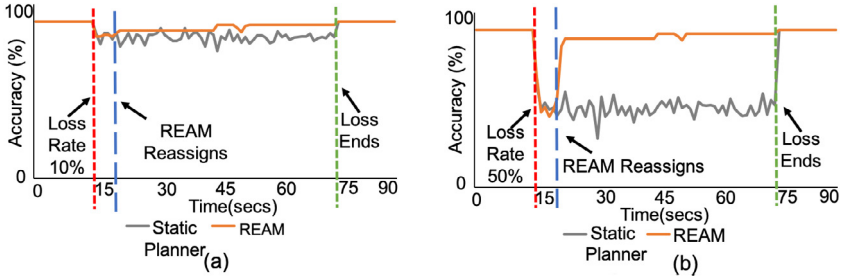

Fig. 15. Average accuracy obtained by the REAM and static planner at the network loss rates of: (a) 0.1 and (b) 0.5, respectively.

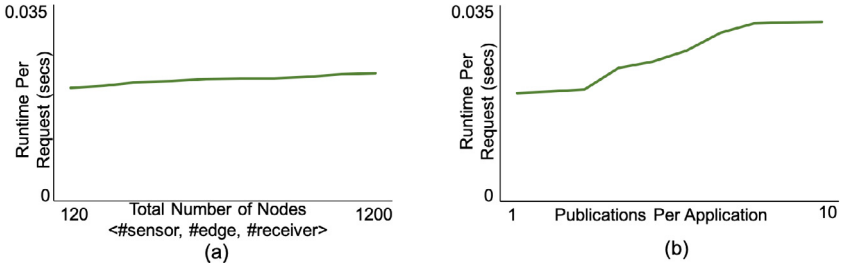

Fig. 16. Running time taken per application in environments raging: (a) from 120 to 1200 nodes and (b) from 1 to 10 tasks.

by the total number of packets sent for that plan. Fig. 15 shows the accuracy values per timestamp achieved by both approaches over the entire experiments for both 0.1 and 0.5 network loss rates. We see that before the introduction of the network loss rates, both REAM and the static planner achieve the same high accuracy since they both use the high utility action plan a_2. We also see that the introduction of network loss results in a drop in accuracy for both approaches. However, REAM's agent recomputes the utility values and selects action plan a_1 unlike the static planner that continues with a_2 and thus suffers from lost packets. The average difference in accuracy levels ranges from 11% with 0.1 network loss rate and 42% with 0.5 network loss rate. When the network condition on the affected link recovers, REAM switches back to action plan a_2.

4.5. Scalability of REAM

In this section, we evaluate the scalability of REAM for large-scale community environments consisting of a multitude of sensors, edge servers, receivers, and monitoring applications with varying action plan complexities. We use the same prototype and simulated input setup as in the network loss experiment. We first varied the total number of sensors, edge servers, and receivers, and then measured the time taken by REAM to handle action plan selections for multiple applications. Fig. 16(a) shows the time taken per application by REAM for 100 applications, in environments ranging from 120 nodes (100 sensors, 10 edge servers, and 10 receivers), to 1200 nodes (1000 sensors, 100 edge servers, and 100 receivers). Each action plan candidate for the applications was set to have 3 tasks (sensing, compute, and publication). We observed that the time taken per application remained relative constant despite the 10x scaling of the environment size. The runtime of REAM ranged from 0.020 to 0.022 s to select and publish action plans for each application which shows a near constant runtime, thus showing REAM's effectiveness in handling large-scale community environments.

We then varied the number of tasks (i.e., the number of data processing steps) associated with each action plan from 1 to 10 for 100 applications. We kept the environment size constant at 1200 (1000 sensors, 100 edge servers, 100 receivers).

Fig. 16(b) shows that there is a small increase in the time taken to select and publish action plans per application while increasing the number of tasks per action plan. The runtime ranges from 0.019 to 0.032 s for an average of 0.027 s per application. Given how small these runtimes are, it is an insignificant increase in the runtime of REAM when handling a larger number of tasks per action plan.

5. Related work

Smart spaces. The concept of continuously monitoring smart spaces has appeared in several research areas, including pervasive computing [26] and ambient sensing [27]. Multiple prior studies [28,29] built smart spaces which were instrumented with in-situ sensors, where raw sensor data were passed through analytics algorithms for semantic interpretation. For example, Cook et al. [30] applied machine learning techniques to predict human activities and generate actions for actuators. Shelke and Aksanli [31] employed six different Machine Learning algorithms, including logistic regression, Naive Bayes, etc., to detect the activities in smart spaces. Barker et al. [32] developed frameworks for non-intrusive occupancy monitoring to minimize power consumption in smart buildings. Venkateswaran et al. [33,34] developed a prioritized impact driven methodology to determine sensor placement in smart water networks. The solution proposed in the current paper can be adopted in diverse smart community spaces for resource efficient, and high quality continuous monitoring.

Edge analytics. While resource-heavy analytics are typically offloaded to cloud data centers, sending raw sensor data to potentially far-away data centers can result in longer response times and larger network traffic. Resources closer to the smart spaces, e.g., on edge servers or cloudlets [4], can be leveraged to host analytics for shorter response times. Similar to cloud computing, management of resources on multiple edge servers have been studied. Merlino et al. [35] proposed a middleware to progressively process the sensor data at the edge server, fog nodes, and the cloud server to accelerate the analytics in smart spaces. Hong et al. [3] considered the problem of splitting analytics into smaller pieces to deploy them on heterogeneous fog nodes. Jang et al. [36] studied the problem of leveraging in-situ and edge sensors to optimize mobile applications. Similar to these papers [3,35,36], we also employ edge servers to host analytics; but the current paper focuses on a logically centralized edge server. This edge server, however, can be an abstraction of an edge server cluster, to apply our solution in distributed setups.

Machine learning for smart spaces. Various Machine Learning algorithms have been applied in smart spaces for edge analytics. Zhang et al. [37] tailored a Convolutional Neural Network (CNN) model for activity recognition analytics to fit it on multiple less-powerful edge nodes. Similarly, Hung et al. [38] proposed VideoEdge, a framework to cost-efficiently determine query plans for analyzing video streams. They used dominant demand to identify the best tradeoff between multiple resources and accuracy and identify a pareto band of potential resource configurations. Das et al. [39] studied the problem of activity recognition in smart homes, but focused on detecting abnormal activity patterns of elderly occupants. Their system alerted elderly occupants whose standard activity patterns are learnt, upon observing an abnormal one. Wemlinger and Holder [40] addressed the activity recognition problem across different environments as some Machine Learning models cannot be easily generalized. Their core objective was to make sure trained classifiers also work well in an environment different from the one used for training. Venkatesh et al. [41] broke the smart health-related analytics into multiple stages. Through identifying common stages, they proposed to use intermediate data streams to avoid duplicated efforts among various analytics algorithms. Oda et al. [42] pushed a step further by employing deep RL to make actuation decisions in smart spaces.

There were also efforts that apply Machine Learning for decision making under resource constraints. For example, Vaisenberg et al. [43] leveraged Partially Observable Markov Decision Process (POMDP) to control surveillance cameras to record events in resource-constrained smart spaces. In their setup, a few PTZ cameras were the critical resource constraints, and a POMDP-based scheduler was designed to control these cameras. Han et al. [44] used RL to develop a middleware for event identification in community water systems. Mao et al. [45] used RL to develop a task packing framework that handles resource demands from multiple tasks in a compute cluster setting. Chen et al. [46] adopted RL algorithms to migrate edge analytics for better energy efficiency or service quality. Their Q-learning based algorithm was evaluated on a testbed with two edge servers and four mobile devices. Gai et al. [47] considered the analytics deployment problem. They also employed RL algorithms to allocate analytics workload among a larger number of edge servers. While we leverage RL techniques, in this paper we focus on continuous monitoring and determine the granularity levels of individual analytics to minimize system-wide resource consumption while still delivering good quality monitoring results.

6. Conclusion

In this article, we present REAM, a *Resource Efficient Adaptive Monitoring* framework at the edge, that balances the resource requirements and objectives of multiple community monitoring applications in order to provide good quality monitoring of community spaces, while incurring low compute, networking, and energy costs. REAM uses RL agents that determine which sensors and analytics workflows to execute by interacting with the community space and obtaining contextual information about the environment. We evaluate the framework with data from two real-world testbeds and observe that the REAM framework can achieve $>$90% monitoring accuracy while incurring $\sim$50% lower resource consumption costs than static monitoring approaches. We demonstrate that the network awareness capabilities of REAM

during decision making, can result in a 42% improvement in accuracy over network agnostic approaches. We also show that REAM can scale to service large community deployments.

As part of our future work, we intend to investigate the following research directions.

- **Cascading events and resource profiles.** Events occurring in the community space have the potential to trigger other events resulting in a cascade. This inter-relationship between events can be modeled to provide more knowledge to the RL agents to proactively predict and react to the occurrence of related events. We also assume a static resource profile for each action plan that is given in advance. The RL paradigm can handle situations where there is partial knowledge of the state by casting the decision problem as a POMDP [48]. We intend to model these inter-event relationships and develop mechanisms for higher robustness to uncertainty in resource requirements.
- **Optimization across edge servers.** In the current article, we focus on scenarios where edge servers work independently. We intend to work on solutions that cater to optimization and resource sharing for applications distributed across multiple edge servers using the coordinator in future work.
- **Sensor and communication failures.** In practice, sensors can fail for various reasons resulting in lack of measurements and a complete picture of the state of the community space for the monitoring application. This can also happen due to failures in communicating the data from the sensors to the edge servers. We intend to consider failure scenarios as part of the framework to improve error resiliency.
- **Learning action plans.** We assume that the execution graph forming each action plan is pre-defined and known to the RL agent. An interesting question is whether the RL agent, given sensor and analytics association rules, was able to learn which combinations would result in good monitoring results under different contexts thus giving system administrators more freedom to deploy sensors and analytic operators.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by National Science Foundation Award No. CNS1528995 and CNS1952247, U.S. Army Small Business Technology Transfer Program Office and the Army Research Office under Contract No. W911NF-20-P-0027, and a NOVATEK fellowship.

References

[1] Y. Sun, H. Song, et al., Internet of things and big data analytics for smart and connected communities, IEEE Access (2016).
[2] A. Zanella, N. Bui, et al., Internet of Things for smart cities, IEEE Internet Things J. 1 (2014).
[3] H. Hong, P. Tsai, et al., Supporting Internet-of-Things analytics in a fog computing platform, in: CloudCom'17, 2017.
[4] V. Bahl, Cloud 2020: Emergence of micro data centers (cloudlets) for latency sensitive computing, in: Middleware, 2015.
[5] G. D'Angelo, S. Ferretti, V. Ghini, Simulation of the internet of things, in: HPCS 2016, 2016, pp. 1–8.
[6] W. Dong, G. Guan, et al., Mosaic: Towards city scale sensing with mobile sensor networks, in: ICPADS, 2015.
[7] P. Venkateswaran, C.-H. Hsu, S. Mehrotra, N. Venkatasubramanian, REAM: Resource efficient adaptive monitoring of community spaces at the edge using reinforcement learning, in: 2020 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2020, pp. 17–24.
[8] R.-T. Innovations, RTI connext DDS, 2018, Accessed: Apr 16 2018.
[9] E. Siow, T. Tiropanis, W. Hall, Analytics for the internet of things: A survey, ACM Comput. Surv. 51 (4) (2018) 74:1–74:36, http://dx.doi.org/10.1145/3204947, URL: http://doi.acm.org/10.1145/3204947.
[10] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.
[11] G. Dulac-Arnold, R. Evans, et al., Deep reinforcement learning in large discrete action spaces, 2015, arXiv preprint arXiv:1512.07679.
[12] S.P. Singh, T. Jaakkola, et al., Reinforcement learning with soft state aggregation, in: NIPS, 1995.
[13] M.T. Hagan, H.B. Demuth, M.H. Beale, O. De Jesús, Neural Network Design, Vol. 20, Pws Pub., Boston, 1996.
[14] V. Mnih, K. Kavukcuoglu, et al., Playing Atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.
[15] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[16] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Netw. Mach. Learn. 4 (2) (2012) 26–31.
[17] F. Chollet, et al., Keras: The Python Deep Learning Library, Ascl, 2018, pp. ascl–1806.
[18] F. Pedregosa, G. Varoquaux, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (Oct) (2011).
[19] P.W. Li, H.L. Zhao, H.T. Yang, S. Sun, Performance evaluation of transport protocol in data distribution service middleware, in: Advanced Materials Research, Vol. 926, Trans Tech Publ, 2014, pp. 1984–1987.
[20] P. Bellavista, A. Corradi, L. Foschini, A. Pernafini, Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations, in: 2013 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2013, pp. 000377–000383.
[21] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, A. Knoll, OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols, in: 2019 IEEE International Conference on Industrial Technology (ICIT), IEEE, 2019, pp. 955–962.
[22] R.-T. Innovations, 2021, http://community.rti.com/rti-doc/510/ndds/doc/html/api_cpp/index.html.
[23] B. Urbonas, P. Stahre, Stormwater: Best Management Practices and Detention for Water Quality, Drainage, and CSO Management, 1993.
[24] J. Redmon, A. Farhadi, Yolov3: An incremental improvement, 2018, arXiv preprint arXiv:1804.02767.
[25] G. Bradski, A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc., 2008.
[26] M. Ebling, R. Want, Satya revisits "pervasive computing: Vision and challenges", IEEE Pervasive Comput. 16 (3) (2017) 20–23.
[27] E. Massaro, et al., The car as an ambient sensing platform, Proc. IEEE 105 (1) (2017) 3–7.

[28] T. Nef, et al., Evaluation of three state-of-the-art classifiers for recognition of activities of daily living from smart home ambient data, Sensors 15 (2015).
[29] T. Stavropoulos, G. Meditskos, et al., DemaWare2: Integrating sensors, multimedia and semantic analysis for the ambient care of dementia, Pervasive Mob. Comput. 34 (2017).
[30] D. Cook, M. Youngblood, S. Das, A multi-agent approach to controlling a smart environment, in: Designing Smart Homes: The Role of Artificial Intelligence, 2006, pp. 165–182.
[31] S. Shelke, B. Aksanli, Static and dynamic activity detection with ambient sensors in smart spaces, Sensors 19 (2019).
[32] S. Barker, et al., Smart*: An open data set and tools for enabling research in sustainable homes, in: SustKDD, August 111, 2012, p. 108.
[33] P. Venkateswaran, et al., Impact driven sensor placement for leak detection in community water networks, in: International Conference on Cyber-Physical Systems, 2018.
[34] P. Venkateswaran, M.A. Suresh, N. Venkatasubramanian, Augmenting in-situ with mobile sensing for adaptive monitoring of water distribution networks, in: Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, 2019, pp. 151–162.
[35] G. Merlino, R. Dautov, et al., Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware, ACM Trans. Internet Technol. (2019).
[36] M. Jang, H. Lee, et al., SOUL: An edge-cloud system for mobile applications in a sensor-rich world, in: Symposium on Edge Computing (SEC'16), 2016.
[37] S. Zhang, W. Li, et al., Enabling edge intelligence for activity recognition in smart homes, in: MASS, 2018.
[38] C.-C. Hung, et al., Videoedge: Processing camera streams using hierarchical clusters, in: Symposium on Edge Computing (SEC), 2018.
[39] B. Das, et al., One-class classification-based real-time activity error detection in smart homes, IEEE J. Sel. Top. Sign. Proces. 10 (2016).
[40] J. Venkatesh, B. Aksanli, C. Chan, A. Akyurek, T. Rosing, Cross-environment activity recognition using a shared semantic vocabulary, Pervasive Mob. Comput. 51 (2018) 150–159.
[41] J. Venkatesh, B. Aksanli, C. Chan, A. Akyurek, T. Rosing, Modular and personalized smart health application design in a smart city environment, IEEE Internet Things J. 5 (2) (2018) 614–623.
[42] T. Oda, et al., Design of a deep Q-network based simulation system for actuation decision in ambient intelligence, in: AINA, 2019.
[43] R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive Mob. Comput. (2014).
[44] Q. Han, et al., AquaEIS: Middleware support for event identification in community water infrastructures, in: Middleware, 2019.
[45] H. Mao, et al., Resource management with deep reinforcement learning, in: Hotnets, 2016.
[46] M. Chen, et al., A dynamic service migration mechanism in edge cognitive computing, ACM TOIT 19 (2019).
[47] K. Gai, M. Qiu, M. Liu, H. Zhao, Smart resource allocation using reinforcement learning in content-centric cyber-physical systems, in: International Conference on Smart Computing and Communication, Springer, 2017, pp. 39–52.
[48] L.P. Kaelbling, et al., Planning and acting in partially observable stochastic domains, Artificial Intelligence 101 (1998).