REAM: Resource Efficient Adaptive Monitoring of Community Spaces at the Edge Using Reinforcement Learning

Praveen Venkateswaran*, Cheng-Hsin Hsu‡, Sharad Mehrotra*, and Nalini Venkatasubramanian*
* University of California, Irvine ‡ National Tsing Hua University, Taiwan

***Abstract*—An increasing number of community spaces are being instrumented with heterogeneous IoT sensors and actuators that enable continuous monitoring of the surrounding environments. Data streams generated from the devices are analyzed using a range of analytics operators and transformed into meaningful information for community monitoring applications. To ensure high quality results, timely monitoring, and application reliability, we argue that these operators must be hosted at edge servers located in close proximity to the community space. In this paper, we present a *Resource Efficient Adaptive Monitoring* (REAM) framework at the edge that adaptively selects workflows of devices and operators to maintain adequate quality of information for the application at hand while judiciously consuming the limited resources available on edge servers. IoT deployments in community spaces are in a state of continuous flux that are dictated by the nature of activities and events within the space. Since these spaces are complex and change dynamically, and events can take place under different environmental contexts, developing a one-size-fits-all model that works for all types of spaces is infeasible. The REAM framework utilizes deep reinforcement learning agents that learn by interacting with each individual community spaces and take decisions based on the state of the environment in each space and other contextual information. We evaluate our framework on two real-world testbeds in Orange County, USA and NTHU, Taiwan. The evaluation results show that community spaces using REAM can achieve $> 90\%$ monitoring accuracy while incurring $\sim 50\%$ less resource consumption costs compared to existing static monitoring and Machine Learning driven approaches.**

I. Introduction

With the rise in popularity of smart city initiatives, an increasing number of community spaces are being instrumented with off-the-shelf sensors and actuators to be used as parts of various community monitoring applications [22], [27]. These applications can include traffic monitoring, accident detection for emergency response, water network monitoring, and air quality monitoring, among others. The community spaces can cover diverse geographical areas–classrooms, buildings, road intersections, city districts, etc.–and can be instrumented with varying density and heterogeneity of sensors. We note that the measurements and data from the sensors are analyzed using various *analytics*, composed of multiple *operators*. The sensors and analytics used by each monitoring application are dependent on the application's objective. The analytics in this setting, refer to algorithms that analyze and convert incoming sensor measurements into results based on the application's objective. The analytics could range from simple rule-based heuristics to more complex machine learning models. A common approach is to send the sensor data over the Internet to be analyzed by analytics deployed in resource-rich data centers. However, this can result in congested networks, slow response times, and service interruptions since data centers are often located far from the community spaces [10]. Moreover, monitoring applications that are time-sensitive and critical (e.g., flood, fire detection), can suffer from significant performance degradation that can have a large negative impact on the community.

Leveraging the compute resources of *edge servers* like network gateways or dedicated workstations [1], can ensure that network bandwidth is preserved instead of sending continuous data streams from a multitude of sensors to the cloud. Fig. 1 illustrates a sample road intersection instrumented with various sensors. In this space, cameras, motion sensors, moisture sensors and traffic lights are connected to the edge server and power source via wired connections. Other sensors such as turbidity and pH meters are battery powered and wirelessly connected. Remote services like weather forecasts and social media reports can also be used at the edge to provide external information about the community. System administrators or community stakeholders may choose to concurrently execute different monitoring applications and hence activate different sensors and analytics at any given moment.

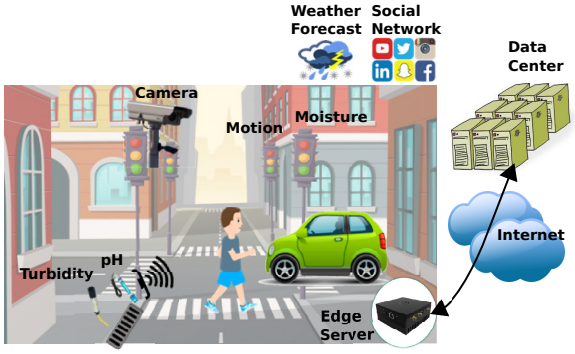

Fig. 1: Sample use case of a community space instrumented with multiple sensors and actuators, an edge server, and other environmental contextual information.

The objectives of monitoring applications could be met using approaches that use different sets of sensors and analytic operators, each of which would require a certain amount of compute, networking and other resources, and would provide a certain quality of results for the application [4], [5]. For

example, a pedestrian detection application could: (1) use video feeds from a surveillance camera and an object detection algorithm, or (2) set up a motion sensor to activate above a certain threshold. The first approach is *fine-grained* and provides more accurate results while incurring much larger costs in terms of compute and networking resources for continuous monitoring than the second approach which is more *coarse-grained*. Since the events driving most monitoring applications are not continuous and can occur sporadically, the costs of utilizing resource-heavy sensors and analytics for continuous monitoring can quickly add up.

Efforts towards monitoring of community spaces using IoT analytics have predominantly been based on the assumption that each application utilizes a specific sensing and analytics approach [27], [21]. However, it is important to note that under certain contexts, coarse-grained approaches can provide sufficient quality results and can also be used to trigger fine-grained approaches. For instance, in the pedestrian detection example described above, the number of instances of pedestrians crossing an intersection on a quiet street during night time would be low. Hence, the coarse-grained motion sensor based approach could be used to detect the potential presence of pedestrians and to then trigger the fine-grained camera based approach if a pedestrian was detected. This *adaptive* approach would be able to achieve sufficient quality of results while incurring lower costs than if the fine-grained approach was run continuously. Community monitoring applications could attain sufficiently accurate results while incurring low costs by intelligently deciding between using different approaches at different times based on the state of the community space and other contextual information. This decision making framework can be implemented in different ways, including simple heuristics, rule-based approaches and learning driven models.

However, community spaces are complex, change dynamically, and events in different spaces can take place under different contexts due to differences in location, demographics, structure, etc., making it 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 conditions. For these reasons, we use Reinforcement Learning (RL) to drive the decision making framework since it employs agents that learn to make better decisions directly from experience by interacting with the environment [23]. In this paper, we study the problem of *Resource Efficient Adaptive Monitoring* of community spaces and present a decision making framework at the edge that dynamically selects the sensors and analytics to execute at any given time to meet the objectives of the monitoring applications. The framework takes into account application priorities while ensuring low compute, networking and energy costs. Specifically, we make the following contributions -

- We present our novel REAM framework deployed at the edge which to our knowledge is among the first resource efficient adaptive monitoring solutions for monitoring community spaces (Section II).
- We formulate the problem and present our reinforcement learning based approach for decision making at the edge (Section III). We concretize the proposed approach using two real monitoring applications: stormwater contamination monitoring and pedestrian counting.
- We evaluate our framework on two real-world testbeds in Orange County, USA and NTHU, Taiwan and compare it to baseline approaches (Sections IV, V).

II. Architecture

In this section, we describe the architecture for our proposed Resource Efficient Adaptive Monitoring (REAM) framework. Fig. 2 illustrates the structure and workflow of the monitoring framework with two edge servers that have three sample monitoring applications running on them.

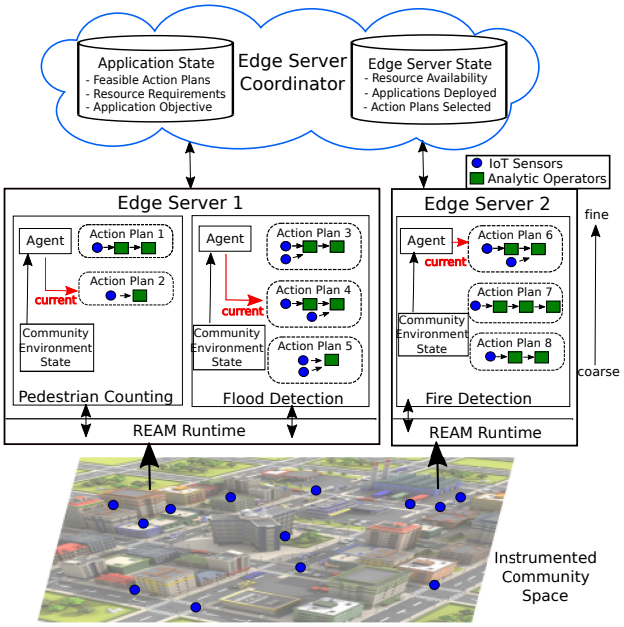

Fig. 2: REAM framework architecture and workflow.

IoT Sensors and Analytic Operators. Each 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 channels like WiFi, Bluetooth, wired connections, ZigBee, LoRa, etc. Once the sensor data is received at the edge server, it is run through a set of analytic operators which could constitute ETL (Extract, Transforming, Load) functions, Machine Learning models, Time-series analyses, among others in order to obtain useful information.

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 an execution graph or workflow of sensors and analytic operators. They can vary in their execution complexity (large workflows with numerous sensor inputs and analytics) and their resource requirements (resource-heavy sensors, large data volumes, complex analytics models). In our framework, as illustrated

in Fig. 2, every monitoring application is a collection of action plans which can be *coarse-grained* and provide 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 than the baseline.

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. 2. The decision of which action plan to execute is taken by a Reinforcement Learning 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, 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. 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 [6]. 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 to create individual agents.

REAM Runtime. Since each edge server has a limited amount of resources, there can be occasions when there are insufficient resources available to execute all the optimal action plans determined by each application's agent. The REAM Runtime is a middleware sitting on each edge server that allocates available resources to each application based on its priority during runtime. The REAM Runtime of an edge server maintains the relative priorities of its monitoring applications by communicating with the Edge Server Coordinator.

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 action plans belonging to each application, their resource requirements as well as the application objectives, and (2) edge server states which include the resource availability at each server and the current applications and action plans deployed. Agents can use this information to take decisions, and system administrators can use the coordinator to modify application objectives and resource availability.

III. 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 a reinforcement learning task, and then present our RL-based approach.

A. Resource Efficient Adaptive Monitoring

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 hosted on a set of edge servers.

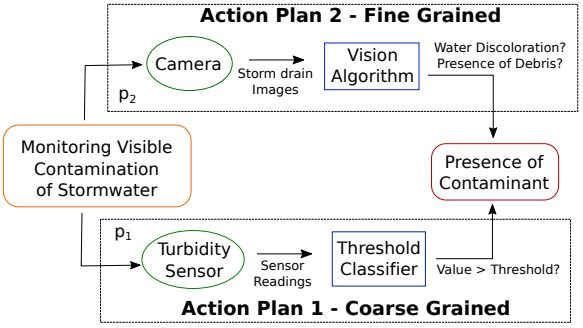

Fig. 3: Example of two action plans for a stormwater visible contamination monitoring application.

We define a set of *action plans* $\mathcal{P}$, where each plan $p_j \in \mathcal{P}$ consists of a workflow of sensors 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 and analytics present in the action plan. We then define the overall utility of an *action plan* p_j as : $\mathcal{U}(p_j) = \frac{\mathcal{B}(p_j)}{\mathcal{C}(p_j)}$.

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 can 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. 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. 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. The REAM agents learn various spatio-temporal characteristics of each community space to provide a customized and accurate monitoring solution.

B. RL Formulation and Prioritized Resource Allocation

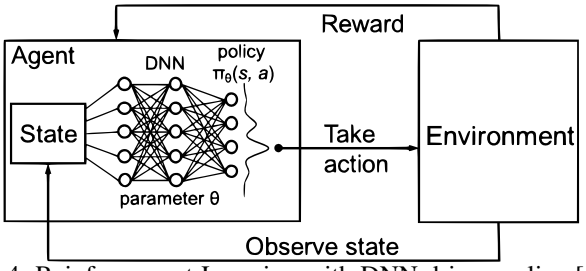

Fig. 4: Reinforcement Learning with DNN driven policy [13].

Consider the general setting shown in Fig. 4, where a reinforcement learning *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, and (4) 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, PTZ for a camera), 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 the 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 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.

Training Algorithm. 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 as well. 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, $Q^*(s, p_j) = E[r_t + \gamma \max Q^*(s_{t+1}, p_{j\,t+1}) | s_t, p_{j\,t}]$ [20].

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. Hence, we use *function approximators* such as Deep Neural Networks (DNNs) [8] to represent the policy by estimating $Q^*(s, p_j)$.

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 this allows 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 [16] as shown in Algorithm 1. It uses an ϵ-greedy policy [23] 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 fixed size and then perform gradient descent to update the weights θ of the neural network using a minibatch of transitions drawn at

random from the buffer.

Prioritized Resource Allocation. The REAM Runtime middleware at each edge server employs our proposed Prioritized Resource Allocation (PRA) algorithm (Algorithm 2) to allocate resources among its different monitoring applications. At each timestep, the REAM Runtime receives action plan change requests from a subset of its monitoring applications' RL agents. It then allocates resources to each application in decreasing order of their priority $\mathcal{A}^{\phi}$. If the edge server does not have sufficient resources available to execute an agent's requested action plan, the agent executes the baseline coarse-grained action plan instead, resulting in potentially lower rewards. This design choice can be easily modified into more comprehensive approaches, for example, to select feasible fine-grained plans.

Algorithm 2 Prioritized Resource Allocation (PRA)

Input: Action plan resource requirements $\mathcal{P}^{\mathcal{R}} = \{p_1^R, ..., p_j^R\}$, Application priority $\mathcal{A}^{\phi}$, Available edge server resources $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_k\}$
for $t = 1 \rightarrow T$ **do**
  Obtain set of action plan requests from subset of agents
  $\mathcal{A}' \subseteq \mathcal{A}$
  Re-order $a_i \in \mathcal{A}'$ based on priority a_i^{ϕ}
  for $i = 1 \rightarrow |\mathcal{A}'|$ **do**
    Obtain resource requirements $p_j^{\mathcal{R}}$ of a_i's chosen action plan p_j
    $\mathcal{R} = \mathcal{R} \setminus p_j^{\mathcal{R}}$
    if $\exists k : \mathcal{R}_k < 0$ **then**
      Allow agent to execute coarse-grained action plan p_0
    else
      Allow agent to execute its chosen action plan p_j
    end if
  end for
end for

IV. Experimental Testbeds

In this section, we describe the real-world community monitoring testbeds located in Orange County, USA and NTHU, Taiwan, and the monitoring applications that we deploy to evaluate the performance of our REAM framework.

Fig. 5: Photos of our testbeds showing: (a) a storm drain and locations of sensing units in Orange County and (b) smart street lamps at NTHU campus.

A. Stormwater Contamination Monitoring

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 (Fig. 5). 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 [24]. 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 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.

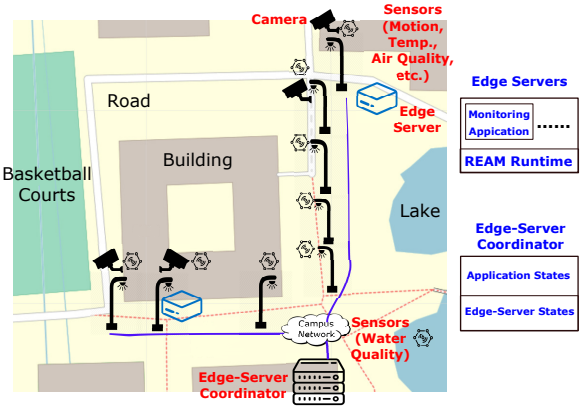

Fig. 6: Sensor-rich smart street lamps - NTHU campus.

B. Pedestrian Counting

We have instrumented eight smart street lamps on the NTHU campus, as shown in Fig. 6, where 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 PTZ camera. 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 resource availability for other on-demand (emergency) applications. Using fine-grained camera feeds coupled with analytic libraries like YOLOv3 [19] and OpenCV [2] 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 REAM is to be able to learn when pedestrians are likely to be present and switch between coarse and fine-grained monitoring to preserve resources.

C. Resource Consumption Measurements

Since the goal of the REAM framework for both the above applications is to be able to achieve application objectives while utilizing as little resources as possible, we capture the actual resource consumption (CPU, networking, power) of the various devices and analytic operators in order to run faithful experiments when comparing our solution against baseline approaches. Table I summarizes the power consumption of the individual devices.

TABLE I: 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	

We also profile the two computer vision libraries (YOLOv3, OpenCV) by analyzing 100 random video frames from the surveillance camera at a resolution of 2048×1536 and 30 frames-per-second. For our setup, YOLOv3 takes 16.28 s to analyze a video frame with an average CPU load of 100%, while OpenCV takes 60.92 ms with an average CPU load of 129%. We note that the measurements are done using CPU only, because GPUs may not be available on edge servers.

V. Evaluations

A. Experimental Setup

Implementation. We implement the REAM framework using Python. The RL agents are implemented using Keras [3], 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 [12] with a learning rate of 0.001 and implement our analytic operators for monitoring applications using Scikit-learn [18].

Data. For the stormwater contamination monitoring application, we use four months of sensor measurements. We use three months for training and one month for testing. The measurements have a granularity of 15 minutes 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.

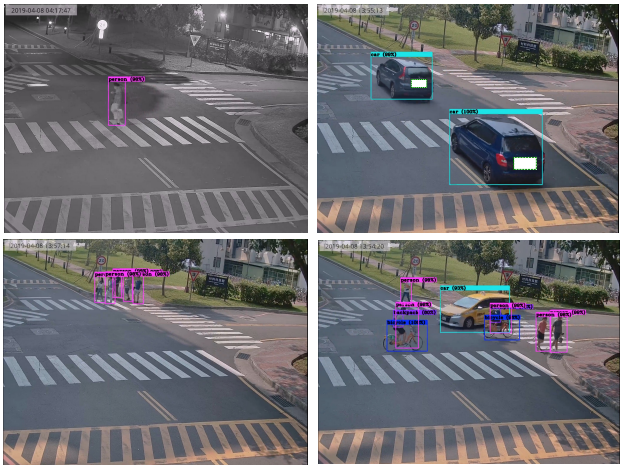

Fig. 7: Sample video frames from a street lamp camera for the pedestrian counting application. The recognized objects and bounding boxes are given by YOLOv3.

For the pedestrian counting application, we use one week's worth of video data from a camera and obtain motion sensor readings for the same period. We use five days for training and two days for testing. The measurements have a granularity of one second. Fig. 7 shows four sample frames of the video data. 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. 7. 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 [2] and YOLOv3 [19] 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 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.

Comparison. For each application, we compare the performance of the REAM framework that adaptively switches between action plans, to the performance of each of the action plans executed in isolation which reflects static monitoring approaches, which is the current practice. We also compare this Reinforcement Learning based framework to a Machine Learning baseline approach utilizing Random Forest that uses the same training and test data to choose action plans. This allows us to determine the effectiveness of the continuous learning provided by the REAM framework with its periodic updates, compared to the supervised learning approach taken by Random Forest.

B. Results

1) Stormwater Contamination Monitoring: We measured the accuracy achieved in classifying contamination events for the test data (Table II) and observed that REAM achieved a 90.9% accuracy which is comparable to the 95.4% achieved by using only the fine-grained action plan and better than the 88.2% obtained by using a Random Forest supervised learning approach and the 73.3% achieved by using just the coarse-grained action plan. Moreover, the REAM framework consumed 44% less energy than the fine-grained action plan, resulting in a longer battery life by 24 days that we derived based on the two D-cell alkaline battery capacity of the In-Situ 600 stormwater sensing unit.

REAM had a 20 minute delay on average in detecting contamination events, compared to the 14, 24 and 45 minute delays achieved by the fine-grained, Random Forest, and coarse-grained approaches respectively. Fig. 8 shows a zoomed in view of the contamination event ground truth and the action plans chosen by the REAM RL agent during a week in 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 III-B 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 minutes on average.

TABLE II: Stormwater Contamination Monitoring

Comparison Approach	Accuracy (%)	Total Energy Consum. (J)	Exp. Batt. Life (days)	Avg. Detection Delay (mins)
REAM	90.9	86.4	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

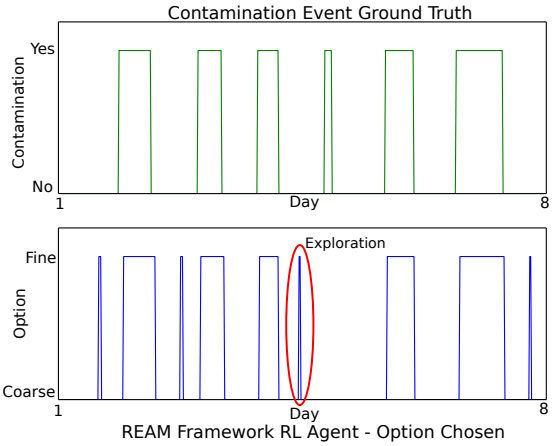

Fig. 8: REAM RL Agent action plan selection to determine contamination events.

TABLE III: Pedestrian Counting

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

2) Pedestrian Counting: Table III shows 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 had a 7.1% error 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. 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% less power consumption over the test period. The REAM framework also results in 40% less data being generated than the YOLOv3 and OpenCV approaches that require continuous generation and transmission of video data.

Fig. 9 illustrates a heatmap based comparison of the pedestrian count ground truth per hour during a week and the corresponding most frequent action plan chosen by the

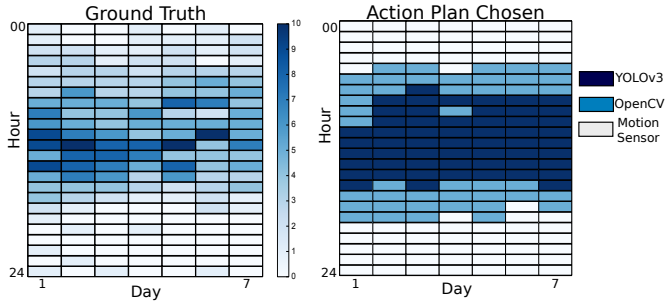

Fig. 9: Comparisons of hourly pedestrian count ground truth and action plan chosen by REAM framework during a week.

REAM RL agent during that hour. We see that the number of pedestrians is the highest during the day (8am - 5pm) and during those periods the predominantly used action plan is YOLOv3 which results in high accuracy. 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.

VI. Related Work and Conclusion

The concept of continuously monitoring smart spaces has appeared in several research areas, including pervasive computing [7] and ambient sensing [14]. Merlino et al. [15] progressively process sensor data at the edge servers, fog nodes, and cloud servers to accelerate the analytics in smart spaces. Hong et al. [10] consider the problem of splitting analytics into smaller pieces to deploy them on heterogeneous fog nodes. Various Machine Learning algorithms have been applied in smart spaces for edge analytics. Zhang et al. [28] tailor a Convolutional Neural Network (CNN) model for activity recognition analytics to fit it on multiple less-powerful edge nodes. Similarly, Hung et al. [11] propose VideoEdge, a framework to cost-efficiently determine query plans for analyzing video streams. There are also efforts that apply machine learning for decision making under resource constraints [26]. Vaisenberg et al. [25] use POMDP to control surveillance cameras to record events in resource-constrained smart spaces. Han et al. [9] and Oda et al. [17] use reinforcement learning for event identification in urban environments. However, these efforts do not cater to heterogeneous sensor inputs, continually changing community environments, and do not focus on resource-efficiency.

In this paper, we present a *Resource Efficient Adaptive Monitoring* (REAM) 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 Reinforcement Learning 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 and also performs better than another Machine Learning approach. Our future work aims to developing resiliency under sensor and communication failures, addressing cascading events by modeling inter-event relationships, and developing learning approaches to automate action plan creation given some sensor and analytics association rules.

References

[1] V. Bahl. Cloud 2020: Emergence of micro data centers (cloudlets) for latency sensitive computing. In *Middleware*, 2015.

[2] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[3] F. Chollet et al. Keras, https://keras.io/, 2015.

[4] G. D'Angelo, S. Ferretti, and V. Ghini. Simulation of the internet of things. In *HPCS 2016*, pages 1–8, 2016.

[5] W. Dong, G. Guan, et al. Mosaic: Towards city scale sensing with mobile sensor networks. In *ICPADS*, 2015.

[6] G. Dulac-Arnold, R. Evans, et al. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[7] M. Ebling and R. Want. Satya revisits "pervasive computing: Vision and challenges". *IEEE Pervasive Computing*, 16(3):20–23, July 2017.

[8] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús. *Neural network design*, volume 20. Pws Pub. Boston, 1996.

[9] Q. Han et al. Aquaeis: Middleware support for event identification in community water infrastructures. In *Middleware*, 2019.

[10] H. Hong, P. Tsai, et al. Supporting Internet-of-Things analytics in a fog computing platform. In *CloudCom'17*, 2017.

[11] C.-C. Hung et al. Videoedge: Processing camera streams using hierarchical clusters. In *Symposium on Edge Computing (SEC)*, 2018.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] H. Mao et al. Resource management with deep reinforcement learning. In *Hotnets*, 2016.

[14] E. Massaro et al. The car as an ambient sensing platform. *Proceedings of the IEEE*, 105(1):3–7, January 2017.

[15] G. Merlino, R. Dautov, et al. Enabling workload engineering in edge, fog, and cloud computing through OpenStack-based middleware. *ACM Transactions on Internet Technology*, April 2019.

[16] V. Mnih, K. Kavukcuoglu, et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[17] T. Oda et al. Design of a deep Q-network based simulation system for actuation decision in ambient intelligence. In *AINA*, 2019.

[18] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct), 2011.

[19] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[20] S. P. Singh, T. Jaakkola, et al. Reinforcement learning with soft state aggregation. In *NIPS*, 1995.

[21] E. Siow, T. Tiropanis, and W. Hall. Analytics for the internet of things: A survey. *ACM Comput. Surv.*, 51(4):74:1–74:36, July 2018.

[22] Y. Sun, H. Song, et al. Internet of things and big data analytics for smart and connected communities. *IEEE access*, 2016.

[23] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[24] B. Urbonas and P. Stahre. *Stormwater: best management practices and detention for water quality, drainage, and CSO management*. 1993.

[25] R. Vaisenberg et al. Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy. *Pervasive and Mobile Computing*, February 2014.

[26] P. Venkateswaran et al. Impact driven sensor placement for leak detection in community water networks. In *International Conference on Cyber-Physical Systems*, 2018.

[27] A. Zanella, N. Bui, et al. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1, 2014.

[28] S. Zhang, W. Li, et al. Enabling edge intelligence for activity recognition in smart homes. In *MASS*, 2018.