Predictive Cyber Foraging for Visual Cloud Computing in Large-scale IoT Systems

Jon Patman, Dmitrii Chemodanov, Prasad Calyam, Kannappan Palaniappan,
Claudio Sterle, Maurizio Boccia

***Abstract*—Cyber foraging has been shown to be especially effective for augmenting low-power Internet-of-Thing (IoT) devices by offloading video processing tasks to nearby edge/cloud computing servers. Factors such as dynamic network conditions, concurrent user access, and limited resource availability, cause offloading decisions that negatively impact overall processing throughput and end-user delays. Moreover, edge/cloud platforms currently offer both Virtual Machine (VM) and serverless computing pricing models, but many existing edge offloading approaches only investigate single VM-based offloading performance. In this paper, we propose a predictive (NP-complete) scheduling-based offloading framework and a heuristic-based counterpart that use machine learning to dynamically decide what combinations of functions or single VM needs to be deployed so that tasks can be efficiently scheduled. We collected over 10,000 network and device traces in a series of realistic experiments relating to a protest crowds incident management application. We then evaluated the practicality of our predictive cyber foraging approach using trace-driven simulations for up to 1000 devices. Our results indicate that predicting single VM offloading costs: (a) leads to near-optimal scheduling in 70% of the cases for service function chaining, and (b) offers a 40% gain in performance over traditional baseline estimation techniques that rely on simple statistics for estimations in the case of single VM-offloading. Considering a series of visual computing offloading scenarios, we also validate our approach benefits of using online versus offline machine learning models for predicting offloading delays.**

***Index Terms*—computation offloading, cyber foraging, edge/cloud computing, machine learning, online job scheduling**

I. Introduction

The recent advances in cloud computing technologies and smart mobile devices have given rise to new systems that bring cloud-like applications and services to users on mobile devices. The maturing Internet of Things (IoT) paradigm has also provided an avenue for offering new video processing services to users on low-power, and often wireless, embedded devices [1]. One of the major challenges for full realization of IoT-based video processing is how best to orchestrate user devices and edge resources that have limited processing capabilities and operate in potentially unreliable networks [2].

As the capability for IoT applications to employ deep neural networks for complex video processing tasks increases, scalability of those services becomes challenging due to device heterogeneity, limited communication and processing capabilities of devices, and dynamic user policies [3]. *Cyber foraging* is a technique that can augment low-power devices with access to more cloud-like services. These services can be leveraged to overcome these limitations by offloading computational workloads from devices to nearby resources at the network edge to reduce application latency and energy consumption of IoT devices [4], [5]. These edge resources known as *cloudlets* are discoverable, generic, stateless nodes located in single-hop proximity and are virtual-machine (VM)-based in order to promote flexibility, mobility, scalability, and elasticity [6].

However, many edge computing solutions to realize cyber foraging rely on consistent network connectivity to the edge/cloud resources, and on strategies that tightly couple mobile clients with servers at deployment time [5], [7]. Such a reliance hinders their adoption for use in critical IoT-based video processing applications such as crisis management in austere network environments [8]. Moreover, with the advent of *Serverless Architectures* [9] that allow delegating managing the execution environment of an application (in the form of *microservice* functions) to the infrastructure provider, there is also a lack of policy-based edge computing solutions capable of both single VM and chain of functions offloading strategies.

In this paper, we propose a *predictive cyber foraging approach that is not only capable of handling edge networks and IoT dynamics, but it also allows different pricing/compute models based on user's policies (e.g., high- vs. low-resolution data) for large-scale IoT-based video processing.* Our approach builds upon our recent prior work [10] and allows for a single VM offloading or a service deterministic estimation processing pipeline. More specifically, our contributions are as follows:

(i) **Data-driven offloading parameters prediction.** Many existing offloading approaches are only evaluated using synthetic data or are only evaluated in small-scale testbeds (e.g., for a single client-server or a few devices). To better understand the complexities involved in real-world multi-edge systems, *we have collected over 10,000 traces publicly available at [11] in a series of realistic* experiments featuring edge hardware and wireless ad-hoc networking in a protest crowd incident management application setting. Specifically, our collected data includes processing and transmission parameters for a computationally-intensive pedestrian and facial recognition pipeline in order to capture ground truth of user workloads being offloaded as a single VM or a chain of functions.

To address the challenges of predicting offloading parameters for various offloading scenarios with different serverless pricing/compute models, we trained and benchmarked the performance of various state-of-the-art offline and online Machine Learning (ML) models. We also gathered insights from these models on performance and operational overhead

This work is supported by the National Science Foundation (NSF) under Award Numbers: CNS-1647182 and CNS-1359125. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

Corresponding author: calyamp@missouri.edu.

trade-offs. We devised several experiments using IoT devices that offloaded low- and high-resolution imagery data streams to nearby cloudlets. Out results show that ML-based predictors outperform traditional network quality estimators with regards to accurately predicting offloading times. We also investigate the performance trade-offs between online and offline learning when: (a) new data becomes available for training, and (b) new data arrives but has a different distribution of values. The performance evaluation results indicate how ML-based prediction can: (a) lead to near-optimal offloading in 70% of cases for service function chaining, and (b) offer a 40% gain in performance when compared to traditional baseline estimation techniques that rely on simple statistics for estimations in the case of single VM-offloading. Our results also show that online learning in our predictive cyber foraging approach can outperform state-of-the-art offline methods in dynamic environments. In particular, we validate the benefits in cases where the online model is allowed to be updated incrementally as new data becomes available. The online model also on average has $\tilde{90}\%$ faster times for both training and prediction over traditional batch-based training.

(ii) **Policy-based edge computing scheme.** We develop a policy-based edge computing scheme for a large amount of IoT devices and edge nodes. Specifically, we have generalized our previous job shop scheduling model for single VM offloading [10] to now be capable of offloading function chains involved in video data processing pipelines. We still seek to minimize the 'maximum schedule time' when all the tasks have finished processing (also known as the *makespan*) across all distributed edge servers. We remark that such an objective minimization allows us to better balance the available physical resources. This in turn increases the acceptance ratio for future offloading requests [12].

However, we found that an optimal solution of our generalized model is intractable and highly complex when considering offloading of few function chains to a couple of edge servers. To address this challenge, we also propose a simple heuristic-based offloading solution for our generalized model. Finally, we perform large-scale trace-driven simulations with up to 1000 devices and edge servers using our real-world datasets. Our results show how our heuristic is able to derive job processing schedules that are well within the upper bounds of the maximum makespan and also demonstrate acceptable optimality performance for offloading function chains.

Paper organization: Section II discusses the related work. Section III describes our proposed predictive cyber foraging approach. Section IV describes collected data and applied ML algorithms, and Section V formulates the online job shop scheduling problem extended to offload function chains. Section VI discusses prediction and offloading results based on trace-driven simulations from our real-world datasets. Lastly, Section VII concludes the paper.

II. Related Work

A. Computation Offloading

Computation offloading is one of the most adopted forms of cyber foraging, and helps in efficient management of resources in edge networks while augmenting the computational capabilities of user devices. The underlying principle in computation offloading involves migrating IoT data and workloads from low-power devices to nearby edge clouds. Heuristic-based algorithms offer negligible overhead for providing estimates, however, they typically perform poorly in dynamic and distributed environments [5]. The recent popular offloading approaches tend to be *data-driven*, meaning they use historical information or domain expertise to model future behavior [13]. In this work, we refer to *predictive* as explicitly using machine learning to not only accurately model target variables but also adapt to changing IoT system environments.

A major challenge of edge computing involves task offloading which is the scheduling of incoming offloading requests from user devices to nearby cloudlets [14], [15]. Task offloading in mobile edge networks is particularly difficult due to the lack of future information about the system and the network instability associated with large-scale IoT networks [16], [3], [10]. Algorithms that schedule offloading decisions based on historical device and network information have been shown to perform significantly better than those that assume a static network model [10], [17], [15]. There has also been much success with modeling offloading problems using heuristics and statistical estimation methods where objectives can be optimizing latency [18], energy consumption [19], [20], and throughput [21], [22].

The ability of offloading frameworks to properly scale for larger networks is essential for the full realization of pervasive edge computing. Several frameworks have been proposed for modeling and optimizing large-scale IoT Networks [14], [23]. However, many existing offloading approaches have practical limitations that are worth considering such as only being evaluated on a small number of devices [24], [25], and often involving simple offloading decisions (i.e., deciding to process locally versus offloading [26], or are concerned with offloading a single service [27]) which may make them difficult to adopt in real world IoT computing environments.

B. Policy-based edge computing

Video processing applications are traditionally designed without decomposability in mind as their code is tightly coupled to the system hardware specifications and software libraries. As a result, most of IoT-based video processing applications find traditional single VM offloading strategy to be the most practical [21]. For this reason, authors in [28] proposed a cost-aware cloud placement for simulated mobile edge computing scenarios. Similarly, the authors in [29] proposed a delay aware policy for the IoT-fog-cloud network to minimize the service delay for applications. These frameworks however, assume that application requests are homogeneous and then allocate the workloads among edge/cloud resources utilizing single VMs to minimize the response time of requests. The work in [30] proposed an application aware workload allocation scheme that is capable of assigning different types of workloads to processing servers as single VMs. All of the previously mentioned frameworks however, do not consider the case where a workload may need to be assigned to a set of functions utilizing multiple resources [31], [32]).

The IoT-based video processing applications can thus benefit from decomposing their functionality into 'microservices' rather than optimizing a single monolithic application. Microservice functions can be executed independently and can communicate with each other to accomplish the original functionality of the application [31], [32], [33]. Commercial services such as Amazon's Rekognition API and AWS Lambda offer related serverless computing capabilities. They however require users to possess hardware expertise for effective use, and also assume that users have reliable network connectivity to centralized cloud data centers. In urban or rural areas experiencing crowds protests or emergency situations [34], applications not only need to operate at the edge of the network infrastructure, but are also prone to operate devices with limited computing resources, intermittent network connectivity, and high levels of stress.

However, a challenge of using serverless computing model involves function decomposition/fusion and function placement. Microservices also bring their own set of unique problems such as increased CPU cycles and network overhead in order to communicate with one another through API calls. Many microservices use the Docker container technology which relies on functions of operating systems to isolate individual containers but in turn can generate more pressure on the operating-system layer [35]. As nodes or network pathways become unreliable, microservices can be deployed within the operating constraints of the available local resources.

As mobile video streaming traffic is anticipated to account for more than 70% of the overall traffic generated by mobile devices by 2019 and may need require computationally-intensive computer-vision analytics, *our goal is to devise a policy-based edge computing approach that can be used by IoT applications for both single VM and function chain computation offloading*. In this context, our work focus is not concerned about comparing which offloading strategy is better as found in [31], [32]. We specifically focus on how to better derive offloading parameters to improve optimality performance of both offloading strategies.

III. Predictive Cyber Foraging Approach

In this paper, we consider a protest crowd incident management case study, where a set of remote devices are incapable of processing data locally and instead offload computation to nearby edge resources in order to provide situational awareness for emergency personnel. The protest crowds incident management case study is depicted in Figure 1 shows how edge resources can be orchestrated in order to provide high-throughput processing for distributed resources in a large geospatial area. Our approach's main components are shown in Figure 2 and consist of: *Analytics*, *Deployment* and *Optimization* engines.

A. Analytics Engine

Application: the main service or chain of services that users are subscribed to. In our protest crowds incident management case study, the application is a visual processing pipeline composed of various image processing and facial recognition operations.

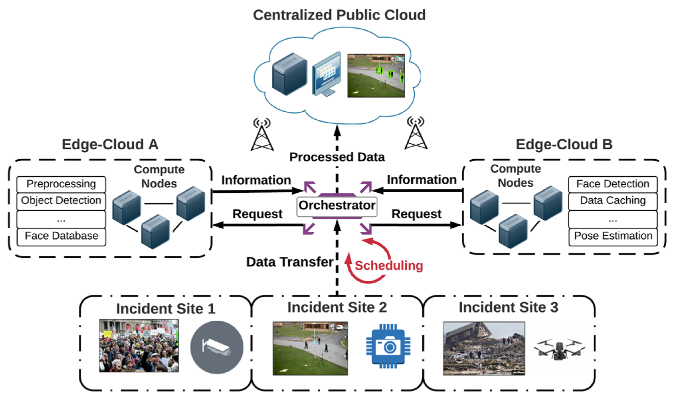


Fig. 1. Conceptual diagram depicting how a protest crowds incident management system can be orchestrated on a set of distributed edge resources. Large numbers of IoT devices upload data via Wi-Fi to nearby access points that are connected by a centralized middleware that can optimally schedule user requests and video data to be processed on available edge hardware. The visual data events can be uploaded to a central cloud via LTE or similar technology and later analyzed by public safety personnel.

Network profiler: captures network information at runtime (e.g., signal strength). The monitoring of network quality is critical for edge networks since poor network performance can cause undesired packet loss and delay between devices and edge resources.

App and device profiler: monitors and collects application/hardware context data asynchronously at runtime. Information such as the size of data to be offloaded, geospatial location, and device battery life are profiled.

Context database: Manages network, device, and context states such as the number of jobs waiting to be processed, the CPU/RAM load for edge nodes, and functions available for processing.

B. Deployment Engine

Virtual Function Instantiation: multiple functions can be deployed on nodes with multi-threading capabilities or can be deployed on nodes determined to be most compatible given the nodes' computing performance.

Container/VM Migration: single VMs and/or functions (Virtual Network Functions (VNFs), microservices) with their corresponding data can be migrated to different parts of the network to better serve geographically distributed IoT device users who may be mobile.

C. Optimization Engine

Predictive Models: information from the network, device profiler, and the context database can be used for training machine learning models with the goal of providing offloading cost predictions that are then used by the runtime scheduler.

Runtime Scheduler: offloading costs from predictive models and meta-data from the context database can be used to derive optimal schedules for batches of incoming IoT video data workloads using combinatorial optimization methods that can be tailored to favor optimality over scalability.

D. Offsite Public Cloud

Remote personnel can also interact with the scheduling middleware by receiving notifications such as the location of

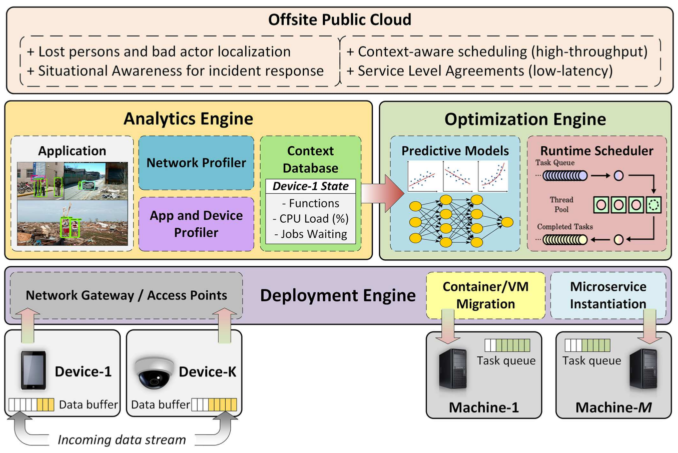


Fig. 2. Predictive cyber foraging approach: as requests arrive from IoT devices at the edge of the network, an analytics engine profiles available network and device resources. Predictive models then make estimates about the associated offloading costs which are then fed into the runtime scheduler in order to derive task schedules and placement of functions. Metadata from these operations can then be uploaded to the cloud for remote interaction.

persons of interest recognized by a facial recognition service. Additionally, scheduling policies can be implemented on the scheduling middleware for diverse edge computing scenarios (e.g., energy-aware and prioritized scheduling). Software-Defined Networking (SDN) can achieve logically centralized control over a distributed network of nodes and mobile devices, thus making SDN a viable option for centralized control of edge computing resources [36]. SDN architectures have also been recently proposed specifically for orchestrating task offloading in mobile edge computing environments [14].

IV. Optimization Engine: Predictive Models

A major challenge in deriving optimal offloading schedules involves estimating resource consumption (e.g., latency, energy consumption, bandwidth utilization) for large amounts of users in dynamic IoT system environments. This challenge is further exacerbated when considering the complexities involved with fluctuations in wireless transmission quality, network congestion, server overburdening, parallel processing capabilities, and scheduling precedence constraints such as those present in microservice chains.

A. *Limitations of Estimation-based Workload Profilers*

Prior to workload placement, information about the data to be processed can be used to estimate execution times. Historical data can allow for accurate modeling but is sensitive to variance and outliers. Furthermore, given the ad-hoc nature of edge computing environments, historical datasets may not be available or feasible to maintain due to e.g., statistical outliers or changing distributions that make it difficult to determine acceptable windows for collecting statistics on historical data. The ability to provide accurate and reliable estimates about workload processing behavior is crucial for achieving efficient schedules especially when prior data is limited.

Static transmission rate estimators can degrade the quality of calculated schedules over time, particularly when relying on measuring the link bitrate or using the *iperf* utility to calculate the transmission time of data [37]. These inaccuracies can prevent scheduling algorithms from making efficient offloading decisions, especially in situations where network and device states are dynamic. Another common issue with traditional estimation techniques is that they are not able to make predictions *a priori*, and therefore need to take measurements periodically. This can lead to increases in network overhead and subsequent delays in scheduling a large number of tasks.

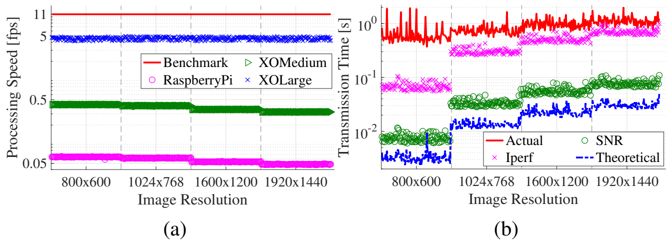


Fig. 3. Experimental analysis for a set of image resolutions relating to: (a) processing speed characteristics for various hardware profiles used in our experiments, and (b) transmission time estimates compared with various measurement estimators such as iperf, Signal-to-Noise Ratio (SNR) that uses the Hartley theorem to calculate the upper-bound bandwidth, and the theoretical minimum.

We collected $\approx 10k$ traces of offloading data available at [11] in a protest crowd incident management application setting. We use this data to evaluate estimation-based versus prediction-based techniques by transferring and processing various image datasets of differing resolution on a variety of ExoGENI testbed [38] nodes arranged in a star topology, as well as a client-server configuration using a Raspberry Pi IoT device. The plots in Figure 3 show the transmission and processing characteristics for a variety of machine types and network conditions collected in our experiments. The benchmark used in Figure 3(a) is from an experiment that used `dlib` for face detection [39] on the same image dataset [40] used in our facial recognition application based on the `dlib` library [41]. As expected, there is a positive correlation between processing speed and hardware type. Additionally, we observe small differences in processing speed between image resolutions as well as a small variance within each image resolution group which suggest that the appearance of faces is not a compounding factor in resulting processing times. In Figure 3(b), each method provides estimates that are much faster than the actual transmission time measured. However, because iperf provides the most accurate estimates over the other techniques we surveyed, we chose to use it as our baseline estimation method. Note that iperf is a lightweight and widely-used tool for estimating network quality [42], [43].

B. *General Learning Problem*

We use Machine Learning to predict the transmission or communication time (t_c) and processing time (t_p) of data to be transmitted and processed by a single VM or a chain of functions and then use these estimates for offloading. We also seek to investigate the prediction performance between different approaches to learning: *incremental* (online) versus *batch* (offline) learning. Generally speaking, batch learning refers to training a predictive model on the entire training dataset, whereas an online algorithm learns from training data that arrive in a sequential stream, without having the entire dataset available from the start. Additionally, online learning can be used as an algorithm that adapts to new data without forgetting its existing knowledge [44]. We focus on regression-based models in this framework due to the fact that continuous-valued metrics used for offloading decision making such as the transmission time or processing time, is a continuous value and not a categorical one.

The general problem of learning can be formulated as follows:
Given a set of training examples $(x_1, y_1), ..., (x_T, y_T)$ where $x_t \in \mathbb{R}^m$ and $y_t \in \mathbb{R}$ (for the regression case) is the target label assigned to x_t. The goal of learning is then to learn a predictive function $F : \mathbb{R}^m \to \mathbb{R}^{|y|}$ where $F(x)$ is a linear function of the form: $F(x) = w^T x$, where $w \in \mathbb{R}$. The performance of the learned function is usually evaluated based on the cumulative penalty suffered by the prediction model. To learn a prediction function that minimizes this penalty over T instances, a loss function denoted as $L(F(x), y)$ is often chosen for minimization. As instances $x_t, t = 1, ..., T$ arrive, a prediction $\hat{y_t}$ is made. This is followed by the environment revealing the true target y_t that is then used as feedback to update the parameters of $F(x)$ [45].

C. Offline Learning

The manner in which machine learning models are trained offline is typically performed via batch learning. This exposes the entire dataset or large portions of the dataset to the model which does not change its approximation of the target function once the initial training phase has begun. Offline learning generally performs quite well but does so at the cost of increased training and inference times. It also leads to increased hardware demands such as memory and processing allocation. Based on our previous results in [10], we have chosen to employ the popular Random Forest algorithm as it has been shown to outperform other contemporary algorithms for predicting offloading costs [26], [37].

Random Forests (RF) are a popular ensemble model belonging to the decision-tree class of ML algorithms. Random forests work by fitting a number of decision trees (100 were used in our experiments) to various subsamples of the dataset that are then averaged to improve accuracy and reduce overfitting in a popular process known as "bagging". Random forests have a variety of useful properties such as the advantage of being able to use the same model for both regression and classification tasks. Random forests also have the ability to learn features that are most important from a collection of features from the training set [46].

D. Online Learning

Online learning algorithms present several benefits over batch learning methods, particularly in the area of computation offloading [26]. Online learning algorithms are also known to scale well to extremely large datasets [47], which makes them desirable candidates for predictive modeling in large-scale IoT serverless edge computing systems.

We employ the popular first-order *Passive-Aggressive* (PA) learning algorithm that makes trade-offs between being: (i) *passive*: preventing the new model deviating too much from the existing one, and (ii) *aggressive*: updating the model by correcting the prediction mistake as much as possible [48]. The optimization of PA learning follows the update rule:

$$w_{t+1} \leftarrow w_t + \frac{\max(0, |y_t - w^T x_t| - \epsilon)}{\|x_t\|^2 + \frac{1}{2C}} \text{sign}(y_t - w^T x_t) x_t \tag{1}$$

where ϵ is the difference between the current prediction and correct value used as a threshold for updating the model, and C is the maximum step size parameter for regularization.

E. Model Asymptotic Complexity

The asymptotic complexity for a Random Forest which is an ensemble of decision trees is as follows. The time complexity for building a random forest is:

$$O(MK\widetilde{N}\log(\widetilde{N}^2)) = O(MKN\log N) \tag{2}$$

where M is the number of randomized trees and K is the number samples required to split a tree node. $\widetilde{N} = 0.632N$ due to the fact that bootstrap samples draw, on average, 63.2% of unique samples [49]. Estimating the depth of a tree is difficult however, as the nodes are expanded until all leaves are pure or until all leaves contain less than K samples. Lastly, the space complexity for the random forest is $O(MlogN)$. The time complexity of making a prediction for a single input sample amounts to the number of operations for traversing each of the M trees, from the root to one of the leaves. Therefore, the time complexity for prediction is $\theta(log(MN))$ for the best and average cases, and is $O(MN)$ in the worst-case.

One of the main advantages of PA-based machine learning is its efficiency, which is basically linear based on the number of training examples. For instance, if X is a matrix of size $(N \times P)$, training has a time complexity of:

$$O(KN\widetilde{P}) \tag{3}$$

where K is the number of iterations (epochs), N is the number of training samples, and $\widetilde{P}$ is the average number of features per sample. PA-based models are therefore well-suited for large-scale machine learning problems.

V. Optimization Engine: Online Scheduler

The mobile edge offloading problem that is used in our protest crowd incident management case study can be formulated as the common online job shop scheduling problem [50]. To this aim, we model mobile IoT devices that request compute offloading *tasks* as a single VM (modeled as a single function) or a chain of *functions* for scheduling, and (mobile) edge servers are modeled as *machines* that process these tasks. The optimization goal is then to schedule each of these tasks onto the appropriate machines in order to finish in the shortest time possible, also known as the minimum makespan [51].

A. Modeling online scheduling

We start our formulation by introducing the following binary variable:

$$x_{jk}^{(i,s)} = \begin{cases} 1, \text{ if task } i \text{ function } s \text{ is assigned} \\ \quad \text{at machine } j \text{ position } k, \\ 0, \text{ otherwise,} \end{cases} \tag{4}$$

where $i \in N$ is a set of independent tasks, $s \in S_i$ is a chain of dependent functions for task i (or can be a single function to model a VM), $j \in M$ is a set of all machines with $k \in N \cdot S$ positions and $S = \max_{i \in N} S_i$.

Having binary variables defined in (4), we start by describing assignment constraints which are essential for the job scheduling problem. The first constraint type ensures that each task function is assigned to exactly one machine position:

$$\sum_{j=1}^{M} \sum_{k=1}^{N \cdot S} x_{jk}^{(i,s)} = 1, \forall i \in N, s \in S_i. \tag{5}$$

Let us introduce another binary variable p_{jk}^m that equals to 1 if exactly m tasks are going to be processed in parallel at machine j position k and 0 otherwise. Thus, the second constraint type ensures that at most $\mathcal{M}$ tasks processing are assigned to each machine position as follows:

$$\begin{gathered} \sum_{i=1}^{N} \sum_{s=1}^{S_i} x_{jk}^{(i,s)} = \sum_{m=0}^{\mathcal{M}} m \cdot p_{jk}^m, \forall j \in M, k \in N \cdot S \\ \sum_{m=0}^{\mathcal{M}} p_{jk}^m = 1, \forall j \in M, k \in N \cdot S. \end{gathered} \tag{6}$$

Note that in general when the edge cloud provider policy $\mathcal{M} \geq 1$, multiple task functions can be processed in parallel at the position k of the machine j.

The third constraint is to ensure that at each machine position we can execute only one function type:

$$x_{jk}^{(i,s)} + x_{j\hat{k}}^{(\hat{i},\hat{s})} \leq 1, \forall i, \hat{i} \in N : i \neq \hat{i}, s, \hat{s} \in S : \hat{s} > s, k \in N \cdot S. \tag{7}$$

Let us also introduce a positive continuous variable y_{jk} that denotes the time when m tasks at position k of machine j will finish being processed. This time should be greater or equal to the sum of the time when machine j finishes processing of the previous position $k-1$ (preceding constraint) and the processing/communication time of m tasks. Thus, we have to satisfy the following set of nonlinear constraints:

$$y_{jk} \geq \left(\sum_{\hat{j}=1}^{M} \sum_{\hat{k}=1}^{k-1} (y_{\hat{j}\hat{k}} + tc_{\hat{j}j}^{is}) \cdot x_{\hat{j}\hat{k}}^{i(s-1)} + \sum_{m=0}^{\mathcal{M}} tp_{ij}^{ms} \cdot p_{jk}^m \right) \cdot x_{jk}^{(i,s)}, \quad \forall i \in N, s \in S, j \in M, k \in N \cdot S, \tag{8}$$

where tp_{ij}^{ms} and $tc_{\hat{j}j}^{is}$ are tasks' processing and communication times.

To avoid non-linearity, we can use a *big M* method, denoted from here as Ω. In order to constrain our optimality conditions within a finite number (typically *big M* is a very large integer value) [52]. Thus, we can generate the following larger set of linear inequalities to substitute (8):

TABLE I
SYMBOLS AND NOTATIONS FOR THE ONLINE OPTIMIZATION PROBLEM

Sets:		
N	$\triangleq$	Set of tasks (offloading mobile devices)
S_i	$\triangleq$	Set of task i functions (e.g., pre-processing, face detection, etc.)
M	$\triangleq$	Set of machines (edge servers with only one VM instance)
Variables:		
$x_{jk}^{(i,s)}$	$\triangleq$	Binary variable that equals to 1 if task i function s is assigned at machine j position k and 0 otherwise
p_{jk}^m	$\triangleq$	Binary variable that equals to 1 if exactly m tasks are going to be processed in parallel at machine j position k and 0 otherwise
y_{jk}	$\triangleq$	Positive continuous variable that denotes the time when m tasks at position k of machine j will finish being processed
α	$\triangleq$	Continuous variable that denotes the maximum makespan
Parameters:		
tp_{ij}^{ms}	$\triangleq$	Task i function s processing time at machine j for m functions processed in parallel
$tc_{\hat{j}j}^{is}$	$\triangleq$	Task i communication (data transfer) time between functions $s-1$ and s and from machine $\hat{j}$ to machine j (or from device to machine j if task $s=1$)
Policies:		
$\mathcal{M}$	$\triangleq$	The maximum number of tasks that can be processed in parallel at the same machine position (note that tasks can be processed in parallel at the same position only if they share the same function)

$$y_{jk} \geq \begin{cases} y_{\hat{j}\hat{k}} + tc_{\hat{j}j}^{is} + \sum_{m=1}^{\mathcal{M}} tp_{ij}^{ms} \cdot p_{jk}^m + \Omega \cdot (x_{\hat{j}\hat{k}}^{i(s-1)} + x_{jk}^{(i,s)} - 2), \\ \qquad \forall i \in N, s \in S, j, \hat{j} \in M, k, \hat{k} \in N \cdot S \\ y_{j(k-1)} + \sum_{m=1}^{\mathcal{M}} tp_{ij}^{ms} \cdot p_{jk}^m + \Omega \cdot (x_{jk}^{is} - 1), \\ \qquad \forall i \in N, s \in S, j \in M, k \in N \cdot S \\ y_{j(k-1)}, \forall j \in M, k \in N \cdot S \end{cases} \tag{9}$$

Note also that $y_{j0} \geq 0$ is a machine offset time - the time when machine j finishes processing the previously scheduled tasks which come online, and $y_{j0} = 0$ for the offline job scheduling problem when all tasks are known in advance.

Let us finally introduce a continuous variable α that denotes the maximum makespan among all machines. Thus, α has to satisfy the following set of constraints:

$$\alpha \geq y_{jN \cdot S}, \forall j \in M. \tag{10}$$

Having both variables and constraints discussed, the online job shop scheduling problem that minimizes the maximum makespan α can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & \alpha \\ \text{subject to} \quad & (5\text{-}7), (9\text{-}10) \\ & x_{jk}^{(i,s)} \in \{0,1\}, \quad \forall i \in N, s \in S, j \in M, k \in N \cdot S \\ & p_{jk}^m \in \{0,1\}, \quad \forall m \in \{0..\mathcal{M}\}, j \in M, k \in N \cdot S \\ & y_{jk} \geq 0, \quad \forall j \in M, k \in N \cdot S, \\ & \alpha \geq 0 \end{aligned} \tag{11}$$

where all variables, parameters and sets are summarized in Table I. Note that if we use a single VM offloading for all IoT-based video processing tasks, the above job shop scheduling problem formulation can be simplified which allows its near-optimal solution in practical time for mid-scale offloading scenarios. The complete reduced formulation can be found in our prior work [10].

B. Heuristic-based online scheduling

To schedule all task functions based on user policies for VM-based or serverless computing at the edge cloud, we can directly solve Problem (11) by using an optimization software package such as CPLEX [53]. However, as we show in Section V, this problem is NP-hard to solve - even for small scale offloading settings it takes several hours on average.

Algorithm 1 Greedy Serverless Edge Offloading

```
Step 1: Init the priority queue $Q$ with first functions of all known tasks based on their estimated maximum makespan using (10) and repeat Steps 2-5 until $Q$ is empty
Step 2: Retrieve and remove the head of $Q$ - function $s$ of task $i$
Step 3: Find the best scheduling position $k$ for tasks $i$ function $s$ based on currently scheduled task functions to minimize the maximum makespan as follows:
    • Find position $k$ of machine $j$ with the minimum of the maximum makespan if function $s$ of tasks $i$ is scheduled as the new function ($\forall j \in M$)
    • Find position $\hat{k}$ of machine $\hat{j}$ with the minimum of the maximum makespan if function $s$ of tasks $i$ qualifies to be consolidated with prior scheduled tasks of type $s$ ($\forall \hat{i} \in N$)
    • Select the best scheduling position based on the maximum makespan between $k$ of machine $j$ and $\hat{k}$ of machine $\hat{j}$
Step 4: Schedule function $s$ of task $i$ at the position and update the corresponding machine makespan
Step 5: Update $Q$ as follows:
    • If exists, estimate the maximum makespan of successive function $s+1$ of tasks $i$ when it is scheduled at the best position as in Step 3 and insert it to $Q$
    • Update all successive functions $s+1$ in $Q$ $\forall \hat{i} \in N$ if consolidated with task $i$
    • Update all successive task functions $\hat{s}+1$ in $Q$ if task functions $\hat{s}$ are scheduled at the right-hand-side position from task $i$ function $s$
```

Thus, we propose a polynomial greedy heuristic outlined in Algorithm 1 for our policy-based edge computing scheme. Our algorithm starts by initializing a queue Q with first functions of each task (e.g., $A_{1..3}$) sorted in ascending order based on their estimated maximum makespan (Step 1) as shown in Figure 4(a). At each iteration until Q is empty, we first retrieve and remove the head of Q (Step 2), then schedule this task i function s (Step 4) at the position that minimizes the maximum makespan across all machines (Step 3), and finally update Q by inserting this task i successive function $s+1$ as well as update all affected task functions in Q (Step 5).

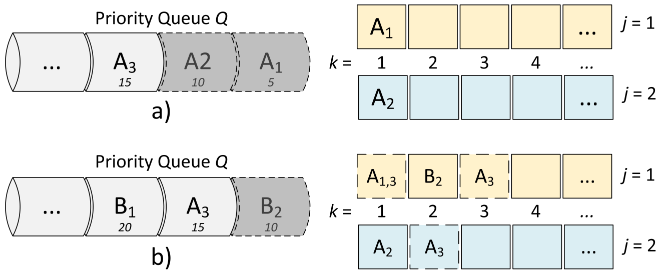


Fig. 4. Heuristic-based scheduling of tasks and functions to various machines. A queue is initialized containing each task sorted in ascending order based on estimated maximum makespan. Each task is removed from the queue and placed at the position that minimizes the makespan across all machines. The queue is finally updated by inserting this task's successive functions and updating the remaining affected task functions in the queue.

For example, if we schedule A_3 to position 1 of machine 1, we need to update the successive functions of A_1 and B_2. The former's successive function needs to be updated in Q, because consolidation of A_1 with A_3 can increase the A_1 completion time. The latter's successive function needs to be updated in Q, because B_2 is located at the right-hand-side position from 1 (i.e., at 2), and thus increasing the completion time of position 1 of machine 1 can increase the completion time of all right-hand-side positions of machine 1 based on (9). We can schedule a task function either at the idle position of some machine or consolidate it with prior scheduled functions if it qualifies. We first describe function consolidation qualification conditions, then follow up with a discussion of the asymptotic complexity of Algorithm 1.

Function consolidation qualification. Function s of task i qualifies for consolidation with other scheduled functions at position k of machine j if the following conditions are met:

- Scheduled task functions at position k of machine j are of the type s
- Number of scheduled task functions at position k of machine j is $< \mathcal{M}$
- None of scheduled task functions at positions $\forall \hat{k} \geq k$ of machine j have their successive functions scheduled at machines $\forall \hat{j} \neq j$.

The first two conditions are necessary to ensure that consolidation of task i function s is feasible. The last condition is needed to ensure that consolidation of task i function s at position k of machine j does not increase the makespan of other machines, i.e., $\forall \hat{j} \in M : \hat{j} \neq j$. For example, consider scheduling of function A task 3 shown in Figure 4(b). Aside from two idle positions $k = 3$ and $k = 2$ of machines 1 and 2, respectively, A_3 can be also consolidated with A_1 at position $k = 1$ of machine 1. This is because A_1 successive function B_1 is not scheduled yet, and B_2 has no successive function. At the same time, we cannot consolidate A_3 with A_2, because A_2 successive function B_2 has been already scheduled. As a result, we can increase the maximum makespan of machine 1, which in turn requires computationally expensive reconstruction of Q.

Asymptotic complexity of Algorithm 1. At any time Q size in Algorithm 1 is at most N. When Q is based on the Fibonacci Heap [54], Step 2 complexity is $O(logN)$. At the same time, Step 3 complexity can be asymptotically bound as $O(N+M)$, and Step 4 complexity is $O(1)$ based on (9). Finally, Step 5 complexity has the following components: the next function insertion to Q is $O(1)$, however we also need to find its best position which has $O(N+M)$ complexity; due to the fact that scheduling of the current task only increases the maximum makespan for all affected functions in Q (at most $N-1$), we can use a decrease key operation which is $O(1)$ complexity. As we need to schedule $N \cdot S$ task functions, the total complexity of our algorithm is:

$$O(NS(\log N + 2(N+M) + N)) = O(NS(\log N + N + M)). \tag{12}$$

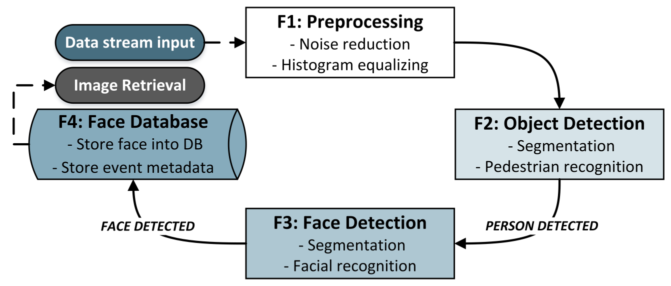


Fig. 5. Face recognition pipeline used in our experiments: the pipeline is either deployed as a single monolithic application (Case I) or as a function chain where each computational function (preprocessing, object detection, face detection, face database) is distributed across multiple resources (Cases II and III).

When $S \ll M$ and $S \ll N$, (12) can be simplified as:

$$O(N\log N + N^2 + NM). \quad (13)$$

We show our algorithm execution time quadratic dependency on the number of tasks and servers in Section VI-D.

VI. Evaluation Results

A. Case Study Experiments

We conducted several experiments to collect realistic data for our protest crowds incident management case study. The experiments resulted in three separate datasets (with a total of ≈ 10k traces publicly available at [11]): *processing*, *transmission-EC* (edge-cloud), and *transmission-AP* (access point), all of which have different input and target features used for training and evaluating our predictive models. The face recognition processing pipeline used in our case study is illustrated in Figure 5. It was deployed in an ExoGENI testbed [38] using a ring topology consisting of 5 nodes with differing computing and storage capabilities [10]. One XOSmall node serves as the client followed by a node dedicated to either a single VM that handles the entire pipeline or to each image processing function (F1-F4) of a chain described in Figure 5.

Images from the "Robust Multi-Person Tracking from Mobile Platforms" dataset were used and consisted of RGB images collected from a camera mounted to a moving vehicle navigating various urban areas with pedestrians present [55]. This dataset was chosen as it is representative of the type of remote images that could be collected in a protest crowds incident management scenario. Moreover, the image dataset contains front-facing pedestrians as well as partially occluded ones which means that for some images, function F2 will detect pedestrians but function F3 will not detect a valid face. This variability allows the predictive models to better generalize for a given set of images instead of learning features correlated with a valid detection in every image.

Table II shows the features that were collected for each dataset during our experiments. The single VM offloading dataset was partitioned into four groups of differing image resolutions (800x600, 1024x768, 1600x1200, 1920x1440), with 100 images per group. For the function chain offloading dataset, we created two subsets of the transmission datasets ($lowres$ and $highres$) of contrasting resolution (640x480, 2048x1536) with 100 images each, that were then transferred

TABLE II
Features collected and used for training models

Transmission Datasets	*Processing* Dataset
Data Attributes:	
Image Height (pixels)	Image Height (pixels)
Image Width (pixels)	Image Width (pixels)
Data Size (bytes)	Data Size (bytes)
-	Number of Persons Detected
-	Number of Faces Detected
Hardware Parameters:	
Signal Level (dB)	Processor Speed (GHz)
Noise Level (dB)	RAM (GB)
Link Quality (%)	Cores per CPU
Bitrate (Mb/s)	Threads per Core
Packet Loss (%)	-
Mean RTT (s)	-
Jitter (s)	-
Mobility:	
Distance (5, 50, & 100 m) ± 15 m	N/A
Prediction Parameters:	
Function Transmission Time (s)	Function Processing Time (s)
Wireless AP Upload Time (s)	Total Processing Time (s)

and processed through all of the functions, with the corresponding transmission and processing times measured. We then collected features related to the networking hardware, processing capabilities of the machines, and attributes of the data being offloaded. Signal level and link quality are both measures of the strength of the connection, while noise, packet loss, and jitter are measures of connection interruptions. Bitrate and mean Round-Trip Time (RTT) are valuable measurements of the capacity and speed of the connection. We also measured processing time on machines with single and multi-threading capabilities in order to represent the parallel processing functionality of executing simultaneous microservices on a single machine.

Processing time was measured for a single VM as well as for each function where the computation and software involved is specific to that particular function. During the *Preprocessing* stage (F1), noise reduction and histogram equalization are performed on images to improve detection capability. In the *Object Detection* stage (F2), images are scanned and detected persons are localized on the image (segmentation) using the Tensorflow's deep learning object detection API. The images are then processed in the *Face Detection* stage (F3) in order to segment faces using the Haar Cascades method available in OpenCV library. Lastly, in the *Face Storage* stage (F4), each face is cropped and stored in a central database where images and meta data such as the location of the image source can be retrieved at a later time.

To measure transmission time for the *Transmission-AP* dataset, we configured a Mikrotik router using the standard mesh routing protocol. A Raspberry Pi served as an IoT-based cloudlet while using a wireless laptop to vary the distance between the client and server at 5, 50, and 100 meters, resulting in 2400 transmission delay samples being collected. Preliminary experiments revealed that distances greater than 100 meters resulted in unstable connections or considerably longer transmission times. For the *Transmission-EC* dataset, we measured the time to transmit the images to each function in our wired ExoGENI testbed where we assume a high-speed connection, therefore delay due to bandwidth is negligible.

B. Comparisons to other methods

To the best of our knowledge, we are the only approach currently focused on using Machine Learning-based techniques to predict offloading costs in order to derive more accurate schedules in a large-scale IoT network. Furthermore, we are the only existing approach to investigate large-scale IoT offloading for more than a single function (i.e. using chains of functions). Implementing other existing approaches for performance comparison has proven difficult as many offloading models and data are incompatible with our study (e.g. they instead focus on optimizing energy consumption [56], content caching [23], or use measurements not collected for our study (e.g. device energy consumption [14], wireless channel frequency [27], [56]). More importantly, a lot of offloading approaches do not contain a *predictive* component, meaning the computation and communication delays are known prior to scheduling, which may not be a practical assumption in a large distributed edge network. One could argue that simple statistics could be used to provide a baseline for initializing these optimizers, however, we counter this argument by pointing out that in a real-world scenario, distributions of incoming requests and network characteristics can change very quickly overtime, rendering statistical analyses obsolete, thus leading to degradation of scheduling optimality.

To illustrate the limitations of such an approach, and provide a comparison between the ML-based estimators with a baseline method, we can use a standard equation for calculating the transmission delay [42], $t_{trans} = L/R_T$, where L is the number of bits of each file and R_T is the transmission rate of the data link measured in bits per second. Calculating the processing time for a given file size on machines with varying hardware specifications, however, proves to be more difficult. Without using historical data, this type of measurement would require knowing how many instructions are executed for each program (which can vary drastically in image processing applications). For this reason, we use the average time, t_{proc}, spent processing a file of L bits on a machine with a given hardware specification.

C. Predicting Offloading Parameters with Machine Learning

1) Model implementation and settings: We have implemented and evaluated a mixture of linear and nonlinear approaches in order to best survey the accuracy of ML models for predicting transmission and processing times. Specifically, we chose Linear Regression, Multi-Layer Perceptron, Support Vector Regression, Decision Tree, k-Nearest Neighbor, Random Forest (RF) and Passive-Aggressive (PA) algorithms as our models. The RF and PA machine learning models are described in more detail in Section IV and the remaining models are thoroughly described in our previous work [37]. All input feature values were normalized, and the models were implemented using the popular Scikit-learn Python library [57].

TABLE III
MACHINE LEARNING PREDICTOR RESULTS (RMSE) FOR CASE I

Model Type	Transmission Time (s) RMSE	Processing Speed (s) RMSE
Baseline Estimate	2.700 ± 2.223	8.285 ± 6.520
Decision Tree	0.877 ± 0.227	**0.231 ± 0.040**
k-Nearest Neighbors	0.826 ± 0.174	0.250 ± 0.041
Linear Regression	0.835 ± 0.139	0.748 ± 0.041
Multi-Layer Perceptron	0.814 ± 0.179	0.307 ± 0.055
Random Forest Regressor	**0.813 ± 0.263**	0.741 ± 0.147
Support Vector Machine	0.837 ± 0.269	0.878 ± 0.079

Because most ML models in Scikit-learn assume that individual features are normally distributed, that is, Gaussian with a mean of zero and unit variance, we standardized our dataset to remove the negative effects for variances orders of magnitude larger than others. We initialize the RF model to have 100 trees in the forest and use two samples as the number for splitting a node. The number of trees were chosen based on the RF initialization suggestions in [57] and the experimental study in [58] that found large performance increases going from 10 to 250 trees, but found minimal improvements as the number of trees increased beyond 250. For the PA model we use the $L2$ regularization metric, with the *huber* loss that is more robust and less sensitive to outliers in the data than the traditional squared-error loss.

2) Single VM offloading scenario: Our first set of results is shown in Table III and is dedicated to a single VM offloading parameters prediction using offline ML algorithms. Most offloading approaches use this type of scenario as it is the simplest in terms of hardware configuration, network overhead, and offloading task complexity. These results show the benefits of using data-driven models for predicting transmission and processing speed over the baseline estimate that used the *iperf* utility, and the facial recognition benchmark. Not only do the ML models on average have a much lower RMSE than the baseline estimate and benchmark, they also have much better precision in terms of the variance observed for predictions. These findings suggest that using data-driven models for predicting offloading costs may lead to improved and more reliable scheduling performance as they are significantly closer to the actual times encountered.

We hypothesize that the RF and Decision Tree models perform the best in our experiments due to several underlying factors inherent in their design. For instance, decision trees have the ability to produce fine-grained decision branches which tend to perform well for modeling complex relationships. For more complicated relationships such as those exhibited in wireless networks, the RF is ideal because it is composed of a collection of decision trees, also known as an ensemble learning method. In this method, each tree's output is used in a voting system to determine the final output of the forest. This voting system, known as *bagging* reduces the bias and variance of the model [46]. Conversely, the large variance and inaccuracy of the baseline estimate can be attributed to the inadequacy in modeling of the non-linear mapping of features for the processing and transmission data sets.

3) Function chain offloading scenario: In this scenario, we aim to predict offloading parameters for a more complicated function chain offloading scenario to reflect the realistic scenario. In our scenario, some computing nodes may be incapable of processing certain kinds of functions, or depending on current load, may be made more efficient by partitioning the workload. Based on the performance results in Table III, the ML-based estimation methods significantly outperform the baseline estimations for the simpler single VM offloading scenario. Thus moving forward, we specifically investigate the performance trade-offs between the offline (RF) and online (PA) machine learning algorithms for the more complex scenarios involved in function chain offloading.

In order to properly evaluate an online model against an offline model, we devised a set of different cases that represent ways in which incoming data streams can be aggregated for training purposes and at what frequency re-training/updates occur. We devised the following cases based that evaluate the different characteristics of each model such as generalization accuracy, robustness, and adaptability:

- *Case I*: All data is available at runtime or that subsequent updates are not needed or feasible. Both the online and offline models are trained on the same dataset prior to runtime.
- *Case II*: The online and offline models are trained before being deployed but the online model is allowed to update based on incoming data streams that may or may not have the same distribution of features.
- *Case III*: The distribution of the incoming data stream's features changes (e.g., nodes go offline/online, new types of data are requesting to be offloaded). The online and offline models are initially trained but as the distribution of features change, the online model is allowed to update on any newly collected data.

Case I is clearly the best-case scenario for the offline model as it guarantees the largest amount of data available for training as long as the number of outliers are small. For Case II, we divided the entire training dataset by half and initially trained each model using the 50% subset of data. The online model was then allowed to update its parameters by two subsequent streams that consisted of 50% splits of the remaining subset of training data. For Case III, both models were initially trained on only $lowres$ data which comprised approximately 50% of the total training dataset. The online model was then allowed to update itself using two subsequent streams that consisted of 50% splits of the remaining training data. The update data here consisted entirely of $highres$ data and thus had a different distribution of features (e.g., data size, processing times), which are not part of the initial model training. For practical purposes, we assume that features from the data (see Table II) used for online training purposes are measurable and labeled prior to training.

Our performance results in Table IV show the trade-offs between online and offline learning approaches for Cases II & III. We evaluate the predictive performance of each model using the Root Mean Squared Error (RMSE) where lower RMSE values indicate better model performance. For Cases II & III, the PA model outperforms the RF model on the processing dataset, except in the case of function F2 (object detection). The reason for this is because - F2 case has the most non-linear behavior and is difficult to predict when sample size is small. However, for the Transmission-AP dataset, the RF model can adequately model the transmission time even with a limited amount of training data but is unable to outperform the PA model when the distribution of features changes over time. Thus, we can conclude that more representative data is needed for the RF model to capture the changing data distribution. For the Transmission-EC dataset, the RF model outperforms the PA model in both cases, owing to the fact that the RF model is able to more accurately model the transmission time even with a limited amount of data.

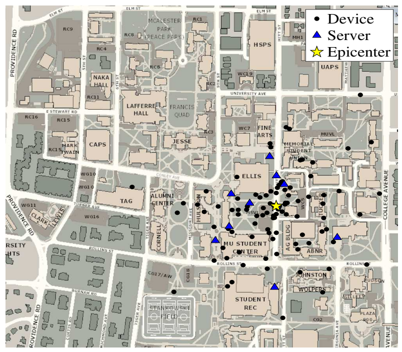
To better understand the operational overhead of using one learning approach over another, we measured the training and prediction times between each model on each of the datasets and report the results in Table V. Even when using the full training dataset (Case I), the PA model outperforms the RF model on each of the datasets by several orders of magnitude. Consequently, this makes the PA model ideal for situations where the distribution of the data is quickly changing and models need to be updated. It is intuitive to suggest that the RF model could just be re-trained from scratch instead of using an online approach. However, as described in Section IV-E, RF have a much larger time complexity. This in turn, makes the choice of RF impractical because it significantly affects scheduling times when trained on larger datasets, which is the typical case encountered in real-world application settings.

TABLE IV
CHAIN OFFLOADING PREDICTION RESULTS (RMSE) FOR CASES II & III

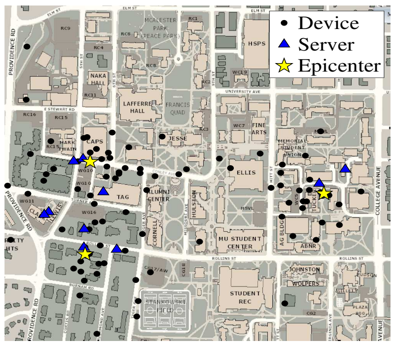
	Case II: Data stream update		Case III: Distribution change	
Function	RF (Offline) RMSE	PA (Online) RMSE	RF (Offline) RMSE	PA (Online) RMSE
Processing Dataset				
F1	0.067 ± 0.034	$\mathbf{0.046 \pm 0.002}$	0.066 ± 0.002	$\mathbf{0.057 \pm 0.004}$
F2	$\mathbf{0.163 \pm 0.011}$	3.512 ± 1.542	$\mathbf{0.154 \pm 0.008}$	0.461 ± 0.033
F3	1.529 ± 0.385	$\mathbf{1.496 \pm 0.271}$	2.824 ± 0.096	$\mathbf{1.956 \pm 0.216}$
F4	0.049 ± 0.025	$\mathbf{0.049 \pm 0.005}$	0.089 ± 0.002	$\mathbf{0.064 \pm 0.005}$
Transmission-AP Dataset				
F1	$\mathbf{0.446 \pm 0.110}$	0.852 ± 0.232	0.658 ± 0.023	$\mathbf{0.387 \pm 0.057}$
F2	$\mathbf{0.452 \pm 0.108}$	0.876 ± 0.232	0.725 ± 0.030	$\mathbf{0.379 \pm 0.049}$
F3	$\mathbf{0.451 \pm 0.109}$	0.816 ± 0.227	0.725 ± 0.031	$\mathbf{0.386 \pm 0.052}$
F4	$\mathbf{0.457 \pm 0.106}$	0.817 ± 0.219	0.672 ± 0.031	$\mathbf{0.389 \pm 0.050}$
Transmission-EC Dataset				
F1 $\rightarrow$ F2	$\mathbf{0.028 \pm 0.003}$	0.724 ± 0.351	$\mathbf{0.041 \pm 0.007}$	0.066 ± 0.011
F2 $\rightarrow$ F3	$\mathbf{0.039 \pm 0.005}$	0.724 ± 0.344	$\mathbf{0.026 \pm 0.008}$	0.065 ± 0.011
F3 $\rightarrow$ F4	$\mathbf{0.048 \pm 0.006}$	0.791 ± 0.386	$\mathbf{0.042 \pm 0.007}$	0.090 ± 0.009

TABLE V
MODEL TRAINING AND PREDICTION TIMES FOR CASE I

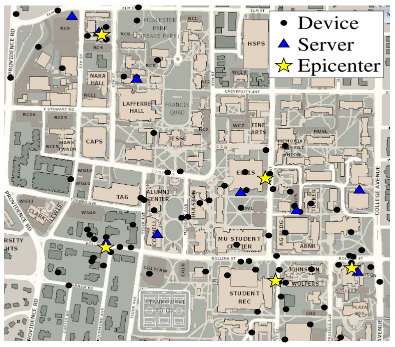
	Training Time (ms)		
ML Model	Processing	Transmission-AP	Transmission-EC
RF	33.33 ± 0.72	31.42 ± 0.46	30.59 ± 0.32
PA	$\mathbf{6.14 \pm 0.73}$	$\mathbf{2.35 \pm 0.10}$	$\mathbf{1.00 \pm 0.00}$
	Prediction Time (ms)		
ML Model	Processing	Transmission-AP	Transmission-EC
RF	3.14 ± 0.49	2.79 ± 0.32	2.32 ± 0.26
PA	$\mathbf{0.25 \pm 0.42}$	$\mathbf{0.30 \pm 0.43}$	$\mathbf{0.20 \pm 0.01}$



(a)



(b)



(c)

Fig. 6. Geospatial maps showing an example of device and server placement during trace-driven simulations for the case of: 1(a), 3(b), and 5(c) Epicenters, respectively. In our protest crowd incident management case study, we represent each protest site as an Epicenter, with a set of heterogeneous resources (servers) available for processing offloading requests from remote devices.

TABLE VI
EXPERIMENT SETTINGS AND PARAMETERS FOR CASES I, II, & III

Simulation Settings	
Experiment Area	1 km^2
Incident Epicenters	1-5
Device-to-Server ratio	10
No. of image offloaded per device	1-10
Device distribution	Normal with $\sigma = \frac{1}{6}$ km
Single VM offloading (Case I)	
No. of devices	50
Resolution range of data	800p, 1024p, 1600p, 1920p
Function Chain Offloading (Case II & Case III)	
No. of devices	1000
Resolution range of data	640p, 2048p

D. *Protest Crowd Incident Management Case Study Results*

In this section, we first describe our results in the case when all application functions are consolidated within a single virtual machine and no parallel thread processing is allowed. In this case, the generalized edge offloading problem in Section V can be significantly simplified (as a job shop scheduling problem) as described in our prior work [10]. We found that estimation inaccuracies can significantly impact scheduling optimality and how machine learning-based predictors can improve the optimality of such scheduling. We then show the complexity of the generalized offloading problem for function chains with allowed parallel processing to give evidence to its current intractability even for small-scale offloading. Additionally, we show how our proposed greedy heuristic algorithm has acceptable (well within the upper bound of the maximum allowable makespan) optimality performance and scalability. Finally, we show optimality trade-offs between offline and online machine learning algorithms for scheduling in different scenarios.

1) General simulation settings: Our Java-based simulation environment is composed of the latest IBM ILOG CPLEX v.12.8 [53] and a High Performance Computing (HPC) Cloud server with two 16-core Intel Xeon Gold 6142 CPUs at 2.6 GHz, 384GB ECC DDR4-2666 RAM running Linux Ubuntu 18.04 STD allocated in the CloudLab infrastructure [59].

Motivated by the challenges of protest crowds incident management described in Section II-B, we generate a 1 km^2 area similar to a typical university campus scale as shown in Figure 6. We then uniformly generate 1 to 5 protest incident epicenters within this area. After that, we place both servers and devices following a normal sampling distribution $\mathcal{N}(\vec{epi}, \sigma^2)$, where the *mean* corresponds to protest epicenter coordinates $\vec{epi}$, and the *standard deviation* $\sigma = \frac{1}{6}$ km (see Table VI for experiment settings and parameters).

In the single VM offloading simulation scenario, we place a total of 50 devices, and use a device-to-server ratio of 10 (unless stated otherwise) to estimate benefits of offloading parameters prediction versus their estimation using mid-scale settings. In the function chain offloading scenario, we use a total of 1000 devices, and use a device-to-server ratio of 10 (unless stated otherwise) to demonstrate our heuristic algorithm performance at large scale settings. Each device also offloads from 1 to 10 images of some resolution: uniformly selected from a set $\{800p, 1024p, 1600p, 1920p\}$ for a single VM offloading, or from a set $\{highres\ (2048p), lowres\ (640p)\}$ for a function chain offloading. We then select a corresponding trace for each device i and server j based on the image resolution, server and device proximity, and server type. We use these traces to obtain ground truth communication tc_{jj}^{is} and processing tp_{ij}^{ms} times (see Table II) as well as use the predicted times from ML algorithms.

After the environment setup, we offload all currently known task functions by taking into account their precedence constraints as well as any parallel processing policies. As a result, task functions can be offloaded as a single VM or a service chain by either consolidating them on one server or by spreading them across many servers in order to minimize the maximum makespan. In this simulation, our main goal is to understand how the prediction/estimation techniques used for offloading parameters affect the optimal scheduling when ground-truth data is used instead. We also aim to evaluate our proposed heuristic performance on function chain offloading settings in large-scale IoT application systems, as well as understand how our proposed online and offline ML algorithms prediction affects the offloading optimality in this case.

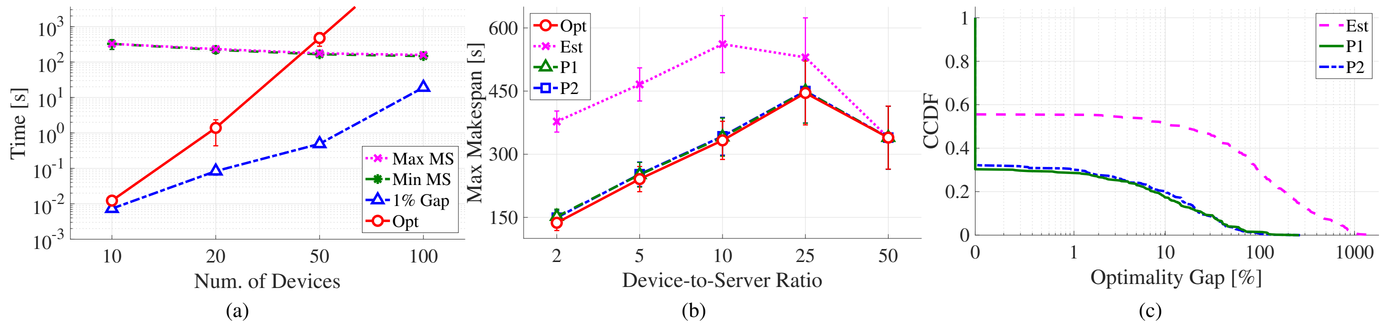


Fig. 7. Evaluation results from trace-driven simulations for single VM offloading comparing the performance between the Optimal, Estimated, Random Forest (P1), and the Multi-Layer Perceptron (P2). (a) Scalability results of Equation 11 for the optimal solution versus 1% optimality gap solutions with respect to the min/max scheduling makespan (MS). (b) Resulting maximum makespan based on a server availability, i.e., device-to-server ratio. (c) Complement Cumulative Distribution Function (CCDF) of the ground-truth optimality gap for baseline estimation versus prediction-based offloading methods.

2) Comparison methods and metrics: We compare the solution to (11) when ground-truth parameters of $tc_{\hat{j}j}^{is}$ and tp_{ij}^{ms} are known in advance (intractable in practice) with its solutions when these parameters are either estimated or predicted. We refer to the ground-truth-based (11) solution as the optimal solution Opt.

To estimate the communication time $tc_{\hat{j}j}^{i1}$ for a single VM offloading, we use the following formula:

$$tc_{\hat{j}j}^{i1} = \frac{\mathrm{data\_size_i}}{\mathrm{throughput_{ij}}}, \tag{14}$$

where the average $throughput_{ij}$ value between device i and server j is estimated using the *iperf* utility. We then estimate the processing time tp_{ij}^{11} for a single VM offloading as follows:

$$tp_{ij}^{11} = \frac{N}{\mathrm{processing\_speed}}, \tag{15}$$

where N is a number of images, and $processing\_speed$ is an average face recognition speed for a specific image resolution obtained from prior benchmark results [39]. We refer to the solution of (11) that is estimated via (14) and (15) parameters as the estimated solution Est.

For a single VM offloading simulation scenario, we then use our offline ML models to predict offloading parameters, and represent the two best pairs of predictors as $(P1)$ and $(P2)$. For predicting single VM offloading parameters, we use the RF model as $(P1)$ and the Multi-Layer Perceptron as $(P2)$, respectively. Finally, for a function chain offloading simulation scenario, we first use ground truth parameters of $tc_{\hat{j}j}^{is}$ and tp_{ij}^{ms} that are known from (11) as evidence of the current intractability of the solution even for small scale offloading settings. Using this same scenario, we also demonstrate how our proposed heuristic has satisfactory results with regard to scalability performance. Subsequently, we predict values of $tc_{\hat{j}j}^{is}$ and tp_{ij}^{ms} with either RF or PA methods to show the impact on offloading optimality (i.e., on the maximum makespan) of offline and online learning, respectively.

The related solutions are compared using *two metrics*: the (11) objective — the maximum makespan α (the lower the better) and its optimality gap measured as a percentage (i.e., the maximum allowable gap between the optimal solution of Problem (11) and the solution obtained by the solver). We measure the optimality gap by substitution of ground-truth offloading parameters to the final schedule produced in order to calculate α (i.e. the maximum makespan achievable). We then estimate the increase in percentage of the ML-based solutions with respect to this new solution, denoted in Figure 7 as Opt. In the general case (service function chain offloading) Equation (11) is hard to solve even for 2 servers. However, in the specific case of the VM-offloading model, the combinatorial complexity reduces significantly, and we can solve this model with 50 servers in practical (near make-span) time.

3) Offloading performance: Our trace-driven simulations produced the following three significant results:

(i) **Predictive cyber foraging approach matches the ground-truth optimal in 70% of cases and has an α value no more than 3 times higher in the worst case.** From our simulations, we found that estimation based edge offloading solutions produce significantly worse scheduling results with respect to both the optimal ground-truth solution and our data-driven approach for a single VM offloading as shown in Figure 7(b). Figure 7(c) shows estimation-based offloading only matches the optimal scheduling in 50% of cases, and can have a maximum makespan over ten times higher in the worst case. In comparison, our predictive approach matches the optimal ground-truth offloading 70% of the time and produces 2-3 times higher maximum makespan in the worst case scenario. Both ML models perform similarly due to the less complex single VM offloading problem (i.e., single function versus a function chain).

$(ii.a)$ **The branch-and-bound-based solution to (11) scales sufficiently for moderate wireless network sizes of up to 100 nodes when only a single VM offloading and a 1% optimality gap policies are allowed.** Our simulations indicate that even when only single VM offloading is allowed, optimally solving (11) is NP-hard and can be intractable for moderate scale networks of 50-100 nodes. To address this intractability, we can allow a 1% optimality gap that enables the branch-and-bound-based solution using IBM CPLEX to scale sufficiently for a large number of nodes. Figure 7(a) shows that for a 100-node edge network, a solution to (11) with a 1% optimality gap can be produced in an order of magnitude less time than either the minimum or maximum makespan requires. Because the optimization problem can be solved in significantly less time, producing solutions will not increase the blocking probability of our online offloading schemes. The relatively shorter scheduling time also allows for the scheduling middleware to begin another batch of tasks while the previous batch is still being processed. Note however that for the case of more than 100 nodes, (polynomial) greedy or approximation algorithms should be used instead.

$(ii.b)$ **The proposed heuristic-based solution of (11) shows satisfactory online performance and scales for large wireless edge network sizes of up to 1000 devices over 1000 edge servers when all policies are allowed.** The plots in Figure 8 show the evaluation results from our trace-driven simulations. In Figure 8(a), the optimal solution computed using (11) is currently intractable even for small offloading settings (i.e., 3 devices). To address this issue, we show the evaluation of our heuristic-based offloading approach in Figure 8(b). Our heuristic is able to outperform by over one order of magnitude the maximum makespan of the produced schedule (necessary for the real-time scheduling) in terms of scheduling time for different device-to-server ratios, and approaches it only for the worst case scenario where only a single device-server pair is considered. Figure 8(c) shows the optimality assessment for our heuristic-based solution with only three devices where we reach $\leq 20\%$ optimality gap for 70% of the cases and 30% of cases where multiple servers are used. We argue that this optimality performance is sufficient due to the overall complexity and intractability of computing the optimal solution for the general problem. Further, this performance is also acceptable when considering the possible scale of multi-edge networks seen in data collection/processing scenarios within the protest incident crowds management case study that we considered for the purposes of this study.

(iii) **Offline for moderate edge network offloading with available historical data, online for large-scale offloading with limited data.** The plots in Figure 9 show the resulting cumulative distribution functions (CDFs) using prediction data generated from Cases I, II, and III described in Section VI-C. Figure 9(a) shows the resulting CDF when using predictions trained on the entire dataset and as expected, both models are much more accurate when the entire dataset is available for training. Another interesting point is that the RF model outperforms the PA model with regard to the actual ground truth measurements when the entire dataset is available.

Conversely, as expected from the results in Table IV, the PA model outperforms the RF model for part of Figure 9(b) (Case II) and all of Figure 9(c) (Case III) although both methods have considerable error with regard to the actual ground truth (GT) measurements when they are not trained on the entire dataset. Therefore, we have to better evaluate the performance between each learning method while also minimizing the error between model predictions and the actual measurements. For this, a much larger dataset would be required so that when models are initially trained, the initial performance is acceptable before model updates are allowed to take place.

VII. Conclusion

In this paper, we proposed a novel predictive cyber foraging approach for providing policy-based computation offloading for visual data processing tasks in a large-scale IoT system. Our novelty is in the consideration of multi-edge network scenarios allowing for a policy-based single VM or function chain offloading strategies. We have collected $\approx 10K$ real-world data traces [11]) using a pedestrian and facial recognition pipeline. We decomposed this computer vision pipeline into microservices to run on a combination of wireless and virtualized hardware. We then benchmarked several state-of-the-art ML models in order to identify the trade-offs between each model, and evaluated the models to find the most appropriate ones for the data processing needs. Specifically, using an in-depth analysis of the trade-offs between offline and online learning mechanisms, we presented robust and adaptable predictions of offloading costs for diverse edge computing cases.

We also investigated the scalability of our approach in a multi-edge system testbed setup by performing a set of trace-driven simulations for a large number of devices and machines. Our analyses of trace-driven simulations indicated that the optimal solution for the general problem is currently intractable when both single VM and function chain offloading strategies are allowed. Thus, we designed a greedy heuristic solution which can achieve satisfactory schedule makespan outputs well within the upper bound of the maximum allowable makespan. We found that the combination of our predictive cyber foraging approach and our heuristic scheduling scheme is a robust and scalable solution for a variety of edge computing scenarios for IoT video data processing.

References

[1] J. Gillis, P. Calyam, A. Bartels, M. Popescu, S. Barnes, J. Doty, D. Higbee, and S. Ahmad, "Panacea's glass: Mobile cloud framework for communication in mass casualty disaster triage," in *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. IEEE, 2015, pp. 128–134.

[2] G. Akpakwu *et. al*, "A survey on 5G networks for the Internet of Things: Communication technologies and challenges," *IEEE Access*, vol. 6, pp. 3619–3647, 2018.

[3] H. Li, *et. al*, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[4] M. Satyanarayanan *et. al*, "Cloudlets: at the leading edge of mobile-cloud convergence," in *2014 6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*. IEEE, 2014, pp. 1–9.

[5] F. Samie *et. al*, "Computation offloading and resource allocation for low-power iot edge devices," in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 2016, pp. 7–12.

[6] M. Satyanarayanan *et. al*, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, 2009.

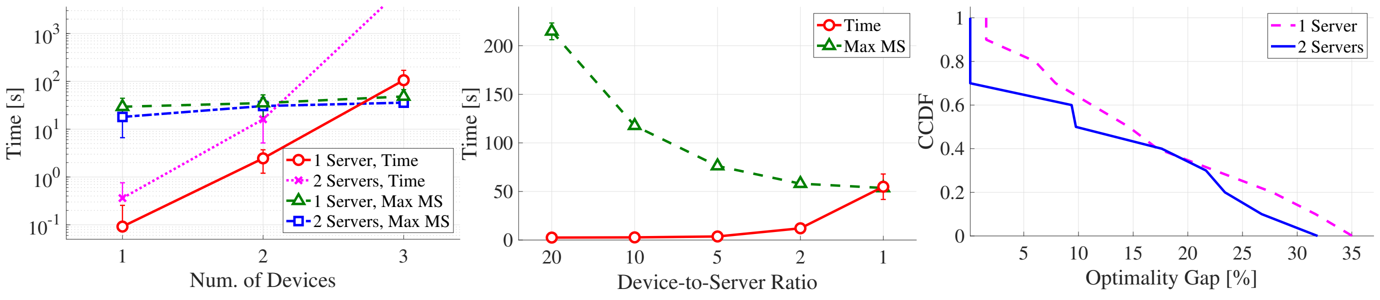
[7] S. Sthapit *et. al*, "Offloading to neighbouring nodes in smart camera network," in *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, 2016, pp. 1823–1827.

[8] G. Lewis *et. al*, "Tactical cloudlets: Moving cloud computing to the edge," in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 1440–1446.

[9] L. Baresi *et. al*, "Empowering low-latency applications through a serverless edge computing architecture," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2017, pp. 196–210.

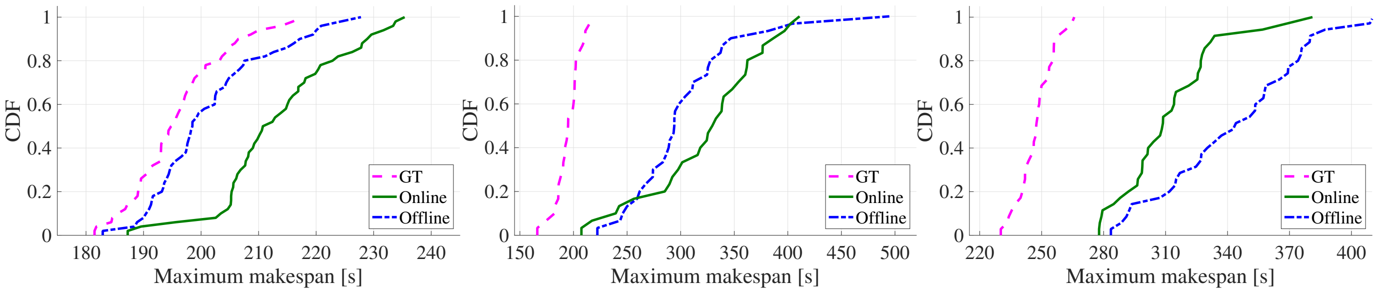
[10] J. Patman *et. al*, "Data-driven edge computing resource scheduling for protest crowds incident management," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018, pp. 1–8.

[11] "Edge computing data," https://bitbucket.org/naas_cloud_computing_project/nm-of-controller/overview, accessed: August, 2019.



(a) (b) (c)

Fig. 8. Trace-driven simulation results for a function chain offloading. (a) Eqn. 11 scalability results for the theoretical optimal solution for up to 2 servers. (b) Scalability results of our heuristic-based solution for 1000 devices and different device-to-server ratios. (c) Complement Cumulative Distribution Function (CCDF) for assessing optimality of our heuristic-based solution for various optimality gaps for up to 3 devices.



(a) (b) (c)

Fig. 9. Cumulative Distribution Functions of the makespans produced using predictions from the online and offline models as well as the Ground Truth (GT) values. (a) Case I: both models are allowed to be trained on the entire training dataset prior to scheduling. (b) Case II: both models are initially trained on half of the training dataset and the online model is allowed to make subsequent updates on the remaining dataset. (c) Case III: both models are trained on the subset of $lowres$ images and the online model is allowed to update on incoming data from the $highres$ dataset.

[12] D. Chemodanov *et. al*, “A constrained shortest path scheme for virtual network service management,” *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 127–142, 2018.

[13] B. Mukherjee, R. L. Neupane, and P. Calyam, “End-to-end iot security middleware for cloud-fog communication,” in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2017, pp. 151–156.

[14] M. Chen and Y. Hao, “Task offloading for mobile edge computing in software defined ultra-dense network,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[15] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, “Edge cloud offloading algorithms: Issues, methods, and perspectives,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–23, 2019.

[16] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, “A deep learning approach for energy efficient computational offloading in mobile edge computing,” *IEEE Access*, vol. 7, pp. 149 623–149 633, 2019.

[17] Z. Liu *et. al*, “Framework for Context-Aware Computation Offloading in Mobile Cloud Computing,” in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, July 2016, pp. 172–177.

[18] M. Alsheikh *et. al*, “Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1996–2018, Fourthquarter 2014.

[19] Q. Ju, *et. al*, “Collaborative in-network processing for internet of battery-less things,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5184–5195, 2019.

[20] Y. Mao, *et. al*, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[21] P. Mach *et. al*, “Mobile Edge Computing: A Survey on Architecture and Computation Offloading,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[22] P. Agrawal, *et. al*, “Energy-aware scheduling of distributed systems,” *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 4, pp. 1163–1175, 2014.

[23] X. Li, *et. al*, “Hierarchical edge caching in device-to-device aided mobile networks: Modeling, optimization, and design,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1768–1785, 2018.

[24] M. Jia, *et. al*, “Qos-aware task offloading in distributed cloudlets with virtual network function services,” in *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, 2017, pp. 109–116.

[25] S. Kosta, *et. al*, “Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in *2012 Proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[26] H. Eom *et. al*, “MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training,” in *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, March 2015, pp. 51–60.

[27] Y. Hao, *et. al*, “Smart-edge-cocaco: Ai-enabled smart edge with joint computation, caching, and communication in heterogeneous iot,” *IEEE Network*, vol. 33, no. 2, pp. 58–64, 2019.

[28] Q. Fan *et. al*, “Cost aware cloudlet placement for big data processing at the edge,” in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[29] A. Yousefpour *et. al*, “Fog computing: Towards minimizing delay in the internet of things,” in *2017 IEEE international conference on edge computing (EDGE)*. IEEE, 2017, pp. 17–24.

[30] Q. Fan *et. al*, “Application aware workload allocation for edge computing-based iot,” *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2146–2153, 2018.

[31] R. Gargees *et. al*, “Incident-supporting visual cloud computing utilizing software-defined networking,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 1, pp. 182–197, 2016.

[32] D. Chemodanov *et. al*, “Policy-based function-centric computation offloading for real-time drone video analytics,” in *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2019, pp. 1–6.

[33] D. Chemodanov *et al*, “On qoe-oriented cloud service orchestration for application providers,” *IEEE Transactions on Services Computing*, 2018.

[34] J. Moghaddam *et. al*, “A device-to-device communication based disaster response network,” *IEEE Transactions on Cognitive Communications and Networking*, 2018.

[35] T. Ueda *et. al*, “Workload characterization for microservices,” in *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1–10.

[36] Y. Zhao, *et. al*, “A survey of networking applications applying the software defined networking concept based on machine learning,” *IEEE Access*, vol. 7, pp. 95 385–95 405, 2019.

[37] J. Patman *et. al*, “Predictive Analytics for Fog Computing using Ma-

chine Learning and GENI," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2018.

[38] M. Berman *et. al*, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.

[39] Y. An *et. al*, "CNNs for Face Detection and Recognition," https://github.com/fusio-wu/CS231A_project, 2017.

[40] W. Yongkang *et. al*, "Patch-based Probabilistic Image Quality Assessment for Face Selection and Improved Video-based Face Recognition," in *IEEE Biometrics Workshop, Computer Vision and Pattern Recognition (CVPR) Workshops*. IEEE, June 2011, pp. 81–88.

[41] D. E. King, "Dlib-ml: A Machine Learning Toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[42] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*. Addison Wesley.

[43] R. Durner, *et. al*, "Performance study of dynamic qos management for openflow-enabled sdn switches," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, 2015, pp. 177–182.

[44] J. Kelleher *et. al*, *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT Press, 2015.

[45] C. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[46] W. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[47] J. Ma *et. al*, "Identifying suspicious urls: an application of large-scale online learning," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 681–688.

[48] K. Crammer *et. al*, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, vol. 7, no. Mar, pp. 551–585, 2006.

[49] G. Louppe, "Understanding random forests: From theory to practice," *arXiv preprint arXiv:1407.7502*, 2014.

[50] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[51] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[52] S. I. Gass and C. M. Harris, "Encyclopedia of operations research and management science," *Journal of the Operational Research Society*, vol. 48, no. 7, pp. 759–760, 1997.

[53] IBM, "ILOG CPLEX v.12.8. User's Manual for CPLEX," https://www.ibm.com/products/ilog-cplex-optimization-studio, 2018.

[54] M. Fredman, *et. al*, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[55] A. Ess *et. al*, "A mobile vision system for robust multi-person tracking," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[56] X. Chen, *et. al*, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[57] F. Pedregosa *et. al*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[58] P. Probst and A.-L. Boulesteix, "To tune or not to tune the number of trees in random forest," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6673–6690, 2017.

[59] R. Ricci, *et. al*, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *; login:: the magazine of USENIX & SAGE*, vol. 39, no. 6, pp. 36–38, 2014.

[60] B. Chun *et. al*, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

Jon Patman received his dual-BS degree in Computer Science and Electronics Engineering from Eastern New Mexico University in 2016. He recently received his M.S. in Computer Science from the University of Missouri-Columbia where he worked as a Research Assistant in the Virtualization, Multimedia and Networking (VIMAN) lab. His current research interests involve designing intelligent systems using the synergy between machine learning, cloud computing, and computer vision.

Dmitrii Chemodanov received his both BS and MS degrees in applied Math and Physics from the Samara State Aerospace University, Russia in 2012 and 2014, respectively. He received his PhD degree in Computer Science from the University of Missouri-Columbia in 2019 and is currently employed with Google. His current research interests include distributed and cloud computing, network and service management, and peer-to-peer networks.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Associate Professor in the Department of Electrical Engineering and Computer Science at University of Missouri-Columbia and directs the VIMAN lab. His current research interests include distributed and cloud computing, computer networking, and cyber security. He is a Senior Member of IEEE.

Kannappan Palaniappan received his PhD from the University of Illinois at Urbana-Champaign, and MS and BS degrees in Systems Design Engineering from the University of Waterloo, Canada. He is a Full Professor in Department of Electrical Engineering and Computer Science at the University of Missouri-Columbia. He directs the Computational Imaging and VisAnalysis (CIVA) Lab. He is a Senior Member of IEEE.

Claudio Sterle received his Ph.D. in Computer Science and Automatic Control from University of Naples Federico II in 2009. He was previously with the Inter-university Research Centre on Enterprise Network, Logistics, and Transportation, Montreal, Canada. He is currently Assistant Professor in Operations Research at DIETI, University Federico II of Naples. His current research interests include exact and heuristic solving methods for complex combinatorial and network optimization problems.

Maurizio Boccia received his Ph.D. in Operations Research from the University of Naples Federico II, Naples, Italy, in 2002. He is currently an Associate Professor in the Department of Electrical Engineering and Information Technology, University Federico II of Naples. His current research interests include computational mixed-integer programming, in particular he worked on network location, network design, routing problems and scheduling problems.