Deep Learning Video Analytics Through Online Learning Based Edge Computing

Heting Liu and Guohong Cao, *Fellow, IEEE*

***Abstract*— Video analytics demand intensive computation resources, which means long processing delay when running on mobile devices. Although offloading computation to the cloud can partially solve the problem, transferring videos to the cloud introduces high transmission delay. With mobile edge computing, computation can be offloaded to the nearby edge servers to reduce the delay. However, the computation resources of the edge servers are usually limited and highly dynamic, and then server selection should be adaptive in order to improve the performance of video analytics. Also, frame resolution should be selected to achieve a better tradeoff between accuracy and frame processing rate. In this paper, we study the server resource-aware offloading problem for video analytics, where the goal is to maximize the utility which is a weighted function of accuracy and frame processing rate. The major challenge to solve this problem is the lack of server and network knowledge and the dynamic system environment. To overcome these challenges, we formulate the problem as a contextual Multi-armed Bandit problem, and propose a Bayesian Optimization based online learning algorithm to gradually learn the server status and the optimal solution, and make it adaptable for time-varying environments. Both theoretical analysis and evaluation results demonstrate the superior performance of our proposed algorithm.**

***Index Terms*— Mobile edge computing, video analytics, offloading, online learning, Bayesian optimization.**

I. Introduction

OVER the past years, deep learning has shown great promise to provide intelligent video analytics to applications such as augmented reality, virtual reality and mobile gaming. For example, Convolutional Neural Network (CNN) models can be used to detect objects showing up in the video frames [1], [2], which facilitates the rendering of animations in augmented reality. Deep learning models demand intensive computation resources, which means long delay when running on mobile devices. One way to solve this problem is to upload videos to the cloud and let the powerful cloud server run the deep learning model. However, uploading videos to the cloud incurs excessive data transmission delay, which may not work well for many latency-sensitive video analytics applications.

Manuscript received 11 January 2021; revised 23 October 2021 and 23 January 2022; accepted 28 March 2022. Date of publication 11 April 2022; date of current version 11 October 2022. This work was supported in part by the National Science Foundation under Grant CCF-2125208 and Grant CNS-1815465. The associate editor coordinating the review of this article and approving it for publication was Z. Cai. *(Corresponding author: Heting Liu.)*

The authors are with the Department of Computer Science and Engineering, Pennsylvania State University, Pennsylvania, PA 16802 USA (e-mail: hxl476@psu.edu; gxc27@psu.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2022.3164598.

Digital Object Identifier 10.1109/TWC.2022.3164598

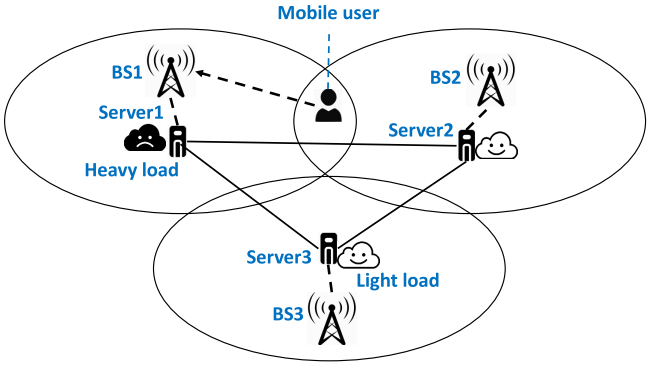


Fig. 1. An example of server selection.

To address this problem, many researchers [3], [4] propose to offload the computation to edge servers which are deployed at the edge of the mobile network such as base stations and access points, to support latency-sensitive applications of nearby users.

However, the computation resources of edge servers are usually limited and may be overloaded when accommodating a large volume of tasks. Offloading computation to an overloaded server may suffer from long processing time and thus degrading the quality of video analytics. Fortunately, users may have multiple choices when offloading their computations [5], [6]. For example in Fig. 1, the mobile user connects to BS1. If server1 is heavily loaded, the mobile user can choose to offload its computation to server2 or server3 where more computation resources are available. Existing work on video analytics focuses on improving performance by selecting different deep learning models or adjusting the video processing configurations such as resolution, frame rate [3], [4], [7], [8], and none of them considers the problem of server selection. Due to the variations of user requests in different regions, the available resources at edge servers are usually different and may change over time. Thus, it is important to make adaptive decisions on which server to offload to achieve better service.

In computation offloading, a typical video analytics application would reduce the frame resolution before transmitting it to the server to save bandwidth and fit the CNN model. Different frame resolutions lead to different accuracy and delays. For example, frame with lower resolution has smaller data size and lower computational cost, and thus has lower transmission

and processing delay. However, using lower resolution also degrades the accuracy. Thus, there is a tradeoff between accuracy and delay when selecting frame resolution, and the resolution selection also depends on the current server resources and network conditions.

In this paper, we study the server resources-aware offloading problem for video analytics, in which we determine where to run the computation (i.e., local device or one of the available edge servers) and using what frame resolution, by addressing the following challenges. First, the user does not have accurate knowledge about the server resources and network condition, and collecting such information incurs extra communication cost. Second, the network condition and server resources are highly dynamic. Moreover, the complex relationship among the accuracy, processing time, frame resolution and server resources, makes it harder to find the optimal resolution through analysis. To overcome these challenges, we formulate the server resource-aware offloading problem as a contextual Multi-armed Bandit (MAB) problem [9], and develop an online learning algorithm based on Bayesian Optimization to gradually learn the server status and the optimal solution only based on local observations. We further propose an adaptive Bayesian Optimization algorithm which can solve the bandit problem in dynamic environments.

The contributions of this paper are summarized as follows:

- We study the server resource-aware offloading problem for video analytics. We formulate it as an optimization problem, where the goal is to maximize the utility which is a weighted function of accuracy and frame processing rate.
- We propose an online learning algorithm based on the Bayesian Optimization framework to select server and resolution using local observations, and make it adaptable for time-varying environments.
- We provide theoretical analysis and regret bound of the proposed algorithm, and conduct extensive evaluations to demonstrate its superior performance.

The rest of paper is organized as follows. Section II reviews related work. In Section III we formulate our server resource-aware offloading problem. We present our online learning based offloading algorithm in Section IV. Section V presents the evaluation results and Section VI concludes the paper.

II. Related Work

Researchers have proposed various offloading frameworks for video analytics in the past several years [7], [8], [10]–[12]. One of the major focuses of these works is how to select the best video processing configurations (e.g., frame resolution, frame sampling rate) to improve the quality of video analytics. For example, Zhang *et al.* [7] proposed VideoStorm which considers the resource and quality tradeoff with multi-dimensional configurations to maximize the performance in terms of quality and delay. Jiang *et al.* [8] presented Chameleon, which can dynamically pick the configuration in a non-stationary environment to balance computational resources consumption and accuracy considering the temporal and spatial correlations of configurations. Besides selecting configurations, Tan and Cao [11], [13] identified the special characteristics of Neural Processing Unit (NPU) and proposed techniques to optimize accuracy or utility of video analytics through edge processing and NPU in mobile device. Zhang *et al.* [10] considered the tradeoff between accuracy and bandwidth consumption of video analytics, and adjusted the data rate to match the available bandwidth while maximizing the achievable accuracy. Tan *et al.* [14] designed resource scheduling algorithms to minimize the processing time for video analytics under memory constraints when using multiple CNN models. Wang *et al.* [12] allocated bandwidth and adjusted the configuration for multiple video streams connected to the same edge node to balance the analytic accuracy and energy consumption while maintaining low latency. However, none of them considers the edge server resources. In mobile edge computing, the server resources are typically limited and the available resources of edge servers vary over time, and then server selection becomes a challenge for edge-assisted video analytics.

Some recent work studied the server selection problem in edge computing [6], [15]–[19]. However, in these works the system information is assumed to be known. For example, Poularakis *et al.* [6] considered a multi-cell environment where users can access multiple servers, and they designed a centralized mobile edge computing operator which knows the load of each server and decides where to execute each request so that the number of served requests is maximized. Tan *et al.* [15] proposed an online job dispatching algorithm to choose a nearby edge server or the remote cloud for offloaded jobs with the aim to minimize the total weighted service response time, where the status of the servers are known. Liu and Cao [19] proposed a reinforcement learning based server selection algorithm to minimize the overall system cost by considering the communication, computation, and server switching cost, where the historical system information are known. Different from them, we do not assume any priori knowledge about the server resource and network condition. To overcome the lack of knowledge on server resources and the system environment, in this work, we model the server resource-aware offloading problem as a contextual multi-armed bandit problem and solve it through Bayesian Optimization based online learning. Although recent work [20] used MAB theory to solve the server selection problem, they assume the reward of an action is a linear function of the contexts, which is not suitable for our problem. More importantly, the reward function may change with time in our problem and then the classic algorithms do not work well. Besides, we need not only to select server, but also to select the frame resolution since both affect the performance.

In summary, existing offloading frameworks for video analytics do not consider the resource constraints of the edge server. In this paper, we take limited edge server resources into consideration when selecting servers for video analytics. Although there are some existing server selection algorithms, they make strong assumptions of the server resources and thus may not work well when the assumptions cannot be satisfied. Moreover, these algorithms can not deal with the time-varying

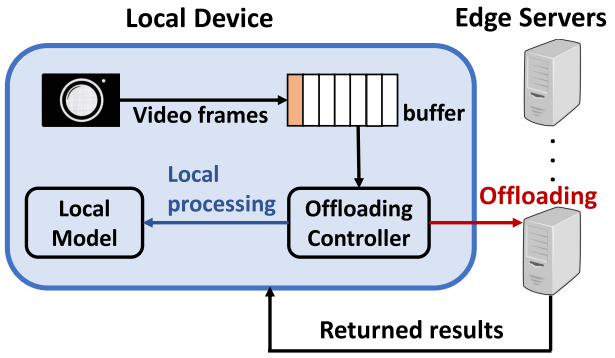


Fig. 2. An overview of the edge-assisted video analytics system.

resources available at the edge server. In this paper, we propose a Bayesian Optimization based online learning algorithm to select server and frame resolution for video analytics in a time-varying environment.

III. System Overview and Problem Formulation

Fig. 2 shows an overview of the edge-assisted video analytics system. Videos are generated locally and sampled with a frame sampling rate f. The frames are first buffered before being analyzed by a CNN model. We adopt the commonly used ResNet-50 CNN architecture [2], [21], which requires the whole frame (image) to be received before being processed. For each buffered frame, the offloading controller decides whether to process it locally or offload it to one of the connected edge servers. If offloading is selected, the offloading controller chooses the edge server and the frame resolution which corresponds to a CNN model. If local processing is selected, the frames are resized to the required resolution of the local CNN model.

The notations used in the problem formulation are summarized in Table I. For a frame i with resolution r, if it is processed locally, it takes $T_i^p(0,r)$ to process the frame and receive the result. Suppose the time of frame i staying in the buffer is T_i^{buf}, its total delay is $L_i(0,r) = T_i^{buf} + T_i^p(0,r)$. If frame i is processed by server m, it takes $T_i^{trans}(m,r) = \frac{s_i(r)}{B} + T_m^c$ to transmit the frame and $T_i^p(m,r)$ to process the frame. Then, the total delay of frame i is $L_i(m,r) = T_i^{buf} + T_i^{trans}(m,r) + T_i^p(m,r)$. Note that the transmission time is affected by the network bandwidth, propagation latency, and the data size; and the processing time is affected by the server resources and the task computation demands.

Suppose the frame rate is f, the time interval between frames arriving at the device buffer is $\frac{1}{f}$, and then the length of the whole video is $n \cdot \frac{1}{f}$. Let x_i^m and x_i^r denote the server selection indicator and resolution selection indicator, respectively. x_i^m indicates which server (the local device or an edge server) is selected to process frame i. If $x_i^m = 1$, frame i is processed by node m; if $x_i^m = 0$, the frame i will not be processed by node m. x_i^r indicates which resolution frame i is resized to. If $x_i^r = 1$, the frame i is resized to resolution r, and $x_i^r = 0$ otherwise. Since the video frames should be analyzed in real time, each frame should be processed within a time constraint T^{max} and the analytics results that are not returned within T^{max} will be discarded. Let x_i^p denote whether frame i is processed within the time constraint. $x_i^p = 0$ if $\sum_m \sum_{r\in\mathcal{R}} x_i^m x_i^r L_i(m,r) > T^{max}$; and $x_i^p = 1$ if $\sum_m \sum_{r\in\mathcal{R}} x_i^m x_i^r L_i(m,r) \le T^{max}$. The total number of frames that are actually processed is $\sum_{i=1}^n x_i^p$. Then the video is processed at a rate of $\frac{f}{n}\sum_{i=1}^n x_i^p$.

When resizing a video frame to resolution r, the accuracy of the corresponding CNN model is denoted as $A(r)$, which can be profiled offline. The average accuracy of the processed frames is $\sum_{i=1}^n \sum_{r\in\mathcal{R}} x_i^p x_i^r A(r) / \sum_{i=1}^n x_i^p$.

As defined in (1), we aim to maximize the *utility* which is a weighted function between frame processing rate and accuracy, because they are the major factors that affect the quality of video analytics. Then, the server resource-aware offloading problem can be formulated as the following optimization problem:

TABLE I
Notations for Problem Formulation

$\mathcal{R}$	the set of all available resolutions
n	the number of video frames that needs to be processed
f	frame sampling rate (fps)
r	frame resolution
m	server index, where 0 represents the local node, and m ($m \neq 0$) represents an edge server
x_i^m	the server selection indicator for frame i
x_i^r	the resolution selection indicator for frame i
x_i^p	the variable to indicate whether frame i is processed within the time constraint
$A(r)$	the accuracy of the CNN model with input frame in resolution r
$s_i(r)$	the data size of the frame i with resolution r
B	upload network bandwidth
$T_i^{trans}(m,r)$	the transmission delay of frame i with resolution r to server m
T_m^c	the network (propagation) latency between the local device and server m
$T_i^p(m,r)$	the processing time of frame i with resolution r at server m
$L_i(m,r)$	the total delay of frame i when selecting server m
T_i^{buf}	the buffering time of frame i
T^{max}	the time constraint for each frame
α	the tradeoff parameter between frame processing rate and accuracy

$$\max \ \frac{f}{n}\sum_{i=1}^{n} x_i^p + \alpha \frac{\sum_{i=1}^n \sum_{r\in\mathcal{R}} x_i^p x_i^r A(r)}{\sum_{i=1}^n x_i^p} \tag{1}$$

$$\text{s.t.} \begin{cases} x_i^p{=}0, if \ \sum_m \sum_{r\in\mathcal{R}} x_i^m x_i^r L_i(m,r) > T^{max}, & \forall i \\ x_i^p{=}1, if \ \sum_m \sum_{r\in\mathcal{R}} x_i^m x_i^r L_i(m,r) \le T^{max}, & \forall i \end{cases} \tag{2}$$

$$\sum_m x_i^m \le 1, \quad \forall i \tag{3}$$

$$x_i^m \in \{0,1\}, \quad \forall i, m \tag{4}$$

$$\sum_{r\in\mathcal{R}} x_i^r \le 1, \quad \forall i \tag{5}$$

$$x_i^r \in \{0,1\}, \quad \forall i, r \tag{6}$$

TABLE II
NOTATIONS FOR ONLINE LEARNING ALGORITHM

a_t	action taken at time t
c_t	context for action a_t
y_t	reward function at time t
$R(T)$	cumulative regret after T rounds
z_t	action-context pair (a_t, c_t)
ϵ_t	zero mean random noise
β_t	exploitation-exploration weight
$k_z(\cdot,\cdot)$	covariance (kernel) function for GP
$k_s(\cdot,\cdot)$	temporal covariance (kernel) function
$\mathcal{D}_t$	collected observations at round t
$\mu_t(\cdot)$	mean of the posterior distribution at round t
$\boldsymbol{k}_t(\cdot,\cdot)$	covariance of the posterior distribution at round t
$\sigma_t^2(\cdot)$	variance of the posterior distribution at round t

Parameter α controls the weight between frame processing rate and accuracy. The optimization variables are x_i^m and x_i^r. Constraint (2) specifies that only frames whose results are received within the time constraint are counted as being processed. Each frame can only be offloaded to one server or processed locally, and we constrain the value of x_i^m in (3) and (4). Similarly, one frame can only be resized to a certain resolution, and we constrain the value of x_i^m in (5) and (6).

To solve this problem, we should know the transmission time and the processing time of each frame, which requires the knowledge of server resources and network bandwidth. However, the server resources and the network condition are not observable to the user and are dynamically changing, which makes such a kind of knowledge hard to obtain. Although the user can request this information through periodic communications with the servers, this approach is extremely inefficient due to the high communication cost. Even with such information, the relationship among utility, resolution and server resource is difficult to model, and then it is hard to find an optimal solution through analysis. To address this problem, in the next section, we transform it to a multi-armed bandit problem and propose a learning based approach which gradually learns the server status and the optimal solution only based on local observations.

IV. ONLINE LEARNING BASED OFFLOADING ALGORITHM

In this section, we present our online learning based server resource-aware offloading algorithm for video analytics. We first transform our problem into the contextual MAB problem since MAB is effective in solving sequential decision making problems under uncertainty. We then leverage the Bayesian Optimization technique to find the optimal solution.

A. Contextual Multi-Armed Bandit Framework

In MAB problems, a decision maker has to select a series of actions of unknown rewards with the goal to maximize the total reward over time [22]. The contextual multi-armed bandit problem is an extension to the classic MAB problem, where additional information related to each action is used by the learner to improve the decisions [23].

Our problem can be transformed into a contextual multi-armed bandit problem (MAB) in the following way. We divide the time into discrete time slots of the same length, denoted as $t = 1, 2, \ldots$, and let τ denote the length of a time slot. The offloading controller performs the role of the decision maker, which selects the server (local device or edge server) and the resolution at the beginning of each time slot. Selecting a server and a frame resolution is regarded as an action, denoted as $a_t = (m, r)$ for time slot t. Then during time slot t, all the frames are resized to the selected resolution and sent to the selected server to process. At the beginning of each time slot, the offloading controller can also observe a context c_t for each action. The context for an action $a_t = (m, r)$ can be defined as $c_t = [r, f, T^{max}, \alpha]$. At the beginning, the offloading controller has little knowledge about each action, and has to select actions randomly to explore their performance. The frames are resized to the selected resolution, and are either processed locally or offloaded to the selected edge server. The offloading controller observes a reward at the end of each time slot based on the returned analytics results. Specifically, the reward is defined as a weighted sum of the accuracy and the real frame processing rate within that time slot. Here the local device counts the number of frames whose analytics results are received during time slot t to measure the current frame processing rate, which is denoted by n_t. n_t is determined by the server resources, network condition, and the computation complexity. The average accuracy corresponds to different resolutions is profiled offline and can be determined once the resolution is selected. Then the reward received of time slot t is

$$y_t = \frac{n_t}{\tau} + \alpha A(r), \tag{7}$$

With time going, the offloading controller accumulates knowledge of the actions, and it selects the server and resolution based on the observed rewards in all previous time slots.

If the reward function is known, the optimal action can be simply selected according to:

$$a^* = \underset{a_t \in \mathcal{A}}{\operatorname{argmax}}\, y_t, \tag{8}$$

where $\mathcal{A}$ denotes the action space. However, due to many unknown parameters (e.g., the processing rate of each server), it is generally hard to derive the reward function and thus unable to choose the optimal action. To solve this problem, the offloading controller has to try out different actions to explore their performances and learn the reward function online. At the same time, it exploits actions that are currently proven to be good. One challenge of designing the learning algorithm is to balance the exploration and exploitation so that the cumulative reward can be optimized. Due to the server resources and network conditions updates, the reward function is not static. Thus the older samples may be stale to learn the current function. How to learn the function in a time-varying environment is another challenge.

As a measure of evaluating the decision strategy, for each time slot, we define the regret Δ_t as the difference between the utility of the optimal action and that of the actual selected action a_t: $\Delta_t = y_t^* - y_t$. After T rounds, the cumulative regret

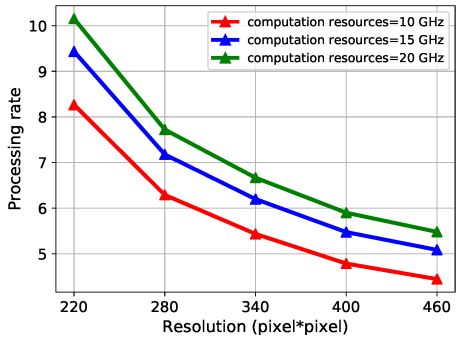


Fig. 3. Processing rate vs. resolution, under different server computation resources.

can be expressed as the following:

$$R(T) = \sum_{t=1}^{T} \Delta_t. \tag{9}$$

An algorithm is called a *no-regret* algorithm if its expected regret is sub-linear in T, i.e., $\frac{R(T)}{T} \rightarrow 0$ for $T \rightarrow \infty$. For example, the existing LinUCB algorithm [23] is known to achieve sub-linear regret under some assumptions. In the classic LinUCB algorithm, it assumes the reward is a linear function of contexts. However, in our problem, because of some nonlinear relations (e.g., the relationship between the processing rate and the resolution as shown in Fig. 3), the reward function is nonlinear, making the LinUCB algorithm not applicable. Besides, LinUCB treats all the previous reward samples as equally important, which may not work well in our problem where the environment is time-varying. In the following subsections, we apply Bayesian Optimization technique to address these challenges.

B. Bayesian Optimization Based Online Learning

Bayesian optimization is a strategy to optimize an unknown function that does not assume any functional forms [24]. In Bayesian optimization, the objective function is treated as a black box and we can only query the function values with arbitrary inputs. Then the function value of a new input is estimated based on the past inputs and function values. In our problem, the objective function to optimize is the reward function as defined in (7), and the input query is an action-context pair. Formally, the reward function y_t is expressed as: $y_t = f(z_t) + \epsilon_t$, where $z_t = (a_t, c_t)$, a_t is the selected action and c_t is the corresponding context. ϵ_t is zero mean random noise: $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$, which is used to measure the deviation between the observed value and the real value.

Since the reward function is unknown, the Bayesian Optimization strategy treats it as a random function and uses a prior distribution that captures beliefs about the behavior of the function. After gathering the function values at some inputs, the prior is updated to form the posterior distribution over the function. The posterior distribution, in turn, is used to determine the next input point. Based on the posterior distribution, we can find which point is more likely to give the optimal function value. Here we define the prior distribution over the objective function as a Gaussian Process (GP) [25], which is the common choice for the prior distribution of Bayesian Optimization. GP is an attractive choice because it is analytical tractable and flexible in modeling nonlinear functions. A $GP(\mu; k)$ can be fully specified by its mean function μ and covariance (or kernel) function k. Let z denote an arbitrary input that is from the set of all action-context pairs Z. The mean function is expressed as $\mu(z) = \mathbb{E}[f(z)]$, which specifies the mean function value at input point z. The covariance function is expressed as $k_z(z, z') = \mathbb{E}[(f(z) - \mu(z))(f(z') - \mu(z'))]$, which specifies the correlation between any pair of input points z and z'. We use the squared exponential kernel as our covariance function, which is a common choice for the covariance function. The squared exponential kernel is defined as

$$k_z(z, z') = exp(-\frac{||z - z'||^2}{2l^2}), \tag{10}$$

where l is called length-scale that determines how much the two points z and z' influence each other. Here the prior over $f(\cdot)$ is taken to be a zero-mean GP with covariance function k as defined in (10).

Suppose the collected observed rewards at round t are $\mathbf{y}_t = [y_1, \ldots, y_t]^T$, with the corresponding inputs $\mathbf{z}_t = [z_1, \ldots, z_t]^T$. When a new point z_{t+1} arrives, the joint distribution of the observations is given by

$$\begin{bmatrix} \mathbf{y}_t \\ y_{t+1} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} \mu(z_{1:t}) \\ \mu(z_{t+1}) \end{bmatrix}, \begin{bmatrix} \mathbf{K}_t + \sigma^2\mathbf{I} & \mathbf{k}_t(z_{t+1}) \\ \mathbf{k}_t^T(z_{t+1}) & k_z(z_{t+1}, z_{t+1}) \end{bmatrix}), \tag{11}$$

where $\mu(z_{1:t}) = [\mu(z_1), \ldots, \mu(z_t)]^T$, $\mathbf{K}_t$ is the kernel matrix $[k_z(z, z')]_{z,z' \in \mathbf{z}_t}$, and $\mathbf{k}_t(z) = [k_z(z_1, z), \ldots, k_z(z_t, z)]^T$. Let $\mathcal{D}_t = \{(z_1, y_1), \ldots, (z_t, y_t)\}$ denote the collected observations at round t. Using the Sherman-Morrison-Woodbury formula [26], [27], we can arrive at the posterior predictive distribution of y_{t+1}:

$$y_{t+1}|\mathcal{D}_t, z_{t+1} \sim \mathcal{N}(\mu(z_{t+1}|\mathcal{D}_t), \sigma^2(z_{t+1}|\mathcal{D}_t)). \tag{12}$$

Thus the posterior distribution over $f(\cdot)$ is a GP again. This is one major computational benefit of GPs that the posterior inference can be performed in closed form. Suppose the mean, covariance and variance of the posterior distribution at round t are denoted as $\mu_t(z)$, $\boldsymbol{k}_t(z, z')$ and $\sigma_t^2(z)$ respectively. These parameters are estimated as

$$\mu_t(z) = \mathbf{k}_t(z)^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{y}_t, \tag{13}$$

$$\boldsymbol{k}_t(z, z') = k_z(z, z') - \mathbf{k}_t(z)^T(\mathbf{K}_t + \sigma^2\mathbf{I})^{-1}\mathbf{k}_t(z'), \tag{14}$$

$$\sigma_t^2(z) = \boldsymbol{k}_t(z, z), \tag{15}$$

where $\mathbf{I}$ is the $t \times t$ identity matrix.

To decide which action to select next, we need to balance the exploitation and exploration. In this work, we adopt the Upper Confidence Bound [23] algorithm based on the above posterior distribution. At time slot t, the algorithm picks action a_t such that

$$a_t = \underset{a_t \in \mathcal{A}}{\operatorname{argmax}}\, \mu_t(a_t, c_t) + \beta_t^{1/2}\sigma_t(a_t, c_t), \tag{16}$$

where β_t is an appropriate constant that controls the exploitation and exploration rate. $\mu_t(\cdot)$ and $\sigma_t(\cdot)$ are posterior mean and standard deviation of the GP conditioned on the observations $\mathcal{D}_t$. This approach implicitly negotiates the exploration-exploitation tradeoff: it exploits actions that achieve high average rewards (large $\mu_t(\cdot)$), and explores actions that are uncertain (large $\sigma_t(\cdot)$) to gather more information.

C. Adaptive Bayesian Optimization

Bayesian optimization based online learning works well when the reward function $f(\cdot)$ is time-invariant. However, the environment may be highly dynamic in practice, resulting in a quick change of the reward function. Thus the samples become increasingly stale with time, and only recent samples have relevant information. The information contained in older samples may be misleading and not effective to learn the current function. To address this problem, we propose an adaptive Bayesian Optimization approach which can solve the bandit problem in dynamic environment.

To adapt the Bayesian Optimization framework to time-varying environments, we extend the reward function to be a function of both action-context pair and time by augmenting the input data with an additional dimension t. Specifically, the input data becomes $\tilde{z} = [t, z]$. To apply GP to the augmented input space, we define a kernel function k_s in the space of time, and then use k_s together with kernel k_z to derive the composite kernel on the product space of time and action-context pairs.

We use the Ornstein-Uhlenbeck temporal covariance function as k_s, that is

$$k_s = (1-\lambda)^{\frac{|t-t'|}{2}}, \quad \lambda \in [0,1]. \tag{17}$$

Then, the product kernel is the composite kernel in the augmented input space, that is

$$\begin{aligned} k(\tilde{z}, \tilde{z}') &= k_s(t,t')k_z(z,z') \\ &= (1-\lambda)^{\frac{|t-t'|}{2}} k_z(z,z'). \end{aligned} \tag{18}$$

From Equation (18), we can see that the correlation of the input data ($\tilde{z}$ and $\tilde{z}'$) is proportional to the similarity of their context, action, and generated time. Since the time intervals are evenly spaced, the new kernel matrix can be modified as $\widetilde{\mathbf{K}}_t = \mathbf{K}_t \circ \mathbf{H}_t$, where $\mathbf{H}_t = [(1-\lambda)^{|i-j|/2}]_{i,j=1}^{t}$. Here, $\circ$ is the Hadamard product. Similarly, we have $\widetilde{\mathbf{k}}_t(z) = \mathbf{k}_t(z) \circ \mathbf{h}_t$ with $\mathbf{h}_t = [(1-\lambda)^{(t+1-i)/2}]_{i=1}^{t}$. In analogy with (13)-(15), the new mean and variance of f_t with respect to the action-context pair z at time t are given by

$$\widetilde{\mu}_t(z) = \widetilde{\mathbf{k}}_t(z)^T (\widetilde{\mathbf{K}}_t + \sigma^2 \mathbf{I})^{-1} \mathbf{y}_t, \tag{19}$$

$$\widetilde{\sigma}_t^2(z) = k(z,z) - \widetilde{\mathbf{k}}_t(z)^T (\widetilde{\mathbf{K}}_t + \sigma^2 \mathbf{I})^{-1} \widetilde{\mathbf{k}}_t(z), \tag{20}$$

Intuitively, the entries in $\mathbf{h}_t$ and $\mathbf{H}_t$ indicate the weight of the corresponding samples in $\mathbf{y}_t$. A stale sample will be assigned less weight, and hence contributes less to the final values of $\widetilde{\mu}_t(z)$ and $\widetilde{\sigma}_t^2(z)$. From the above analysis, we can see that λ is a parameter to control the decreasing rate of the weights of historical samples. With a larger λ, the weights of historical samples decrease faster, i.e., the importance of an "older" sample decreases more than that of a "newer" sample. A larger λ works better for the scenarios where we emphasize the importance of recent samples.

With adaptive Bayesian optimization, the Online Learning based Server resource-aware Offloading algorithm (OLSO) is given in Algorithm 1. At the beginning of each time slot, the controller observes the context for each action, and uses posterior inference to predict the mean and variance for each action-context pair. For each buffered frame, if the time constraint is violated, the offloading controller deletes it from the buffer; otherwise, the offloading controller resizes the frame to the resolution and sends it to the server indicated by the action $a_t = (m, r)$. At line 9, I_t and $I_{t+1} - 1$ denote the first frame and last frame observed in time slot t, respectively. If the result is returned within the time constraint, the number of processed frame of this time slot is increased by 1. The accuracy is set as $A(r)$, which is the average accuracy of the CNN model with image resolution r. At the end of this time slot, the controller calculates the reward as defined in (7). The reward as well as the selected action-context pair z_t are saved to the data set $\mathcal{D}_t$. The running time of OLSO depends on the Bayesian optimization (i.e., (13)-(15)). The Bayesian optimization is performed iteratively as new observations arrive over time which requires $\mathcal{O}(|\mathcal{D}_t|^2)$ running time [28], where $|\mathcal{D}_t|$ denotes the number of data samples in $\mathcal{D}_t$ at time slot t. The algorithm has a time complexity of $\mathcal{O}(R(M+1)|\mathcal{D}_t|^2)$, where R is the total number of resolutions, and M is the total number of servers. In practice, since the maximum size of $\mathcal{D}_t$ is limited, the oldest samples will be discarded. The computation efficiency can be improved to $\mathcal{O}(R(M+1)N^2)$, where N is the maximum size of the data set.

D. Regret Analysis

In this subsection, we provide the theoretical regret bound of OLSO. As defined in Equation (9), the regret shows the difference between OLSO and the optimal. Before showing the regret bound, we define the *mutual information*, which is a key quantity governing the regret bounds: $I(\mathbf{y}_T; f) = H(\mathbf{y}_T) - H(\mathbf{y}_T|f)$. $I(\mathbf{y}_T; f)$ quantifies the reduction in uncertainty (measured in terms of differential Shannon entropy [29]) about f achieved by revealing $\mathbf{y}_T$. In our case, the entropy can be computed in closed form: $H(\mathcal{N}(\mu, \mathbf{\Sigma})) = \frac{1}{2} log|2\pi e\mathbf{\Sigma}|$. Referring to [25] we have

$$I(y_T; f) = H(y_T) - \frac{1}{2}\log|2\pi e\sigma^2\mathbf{I}|. \tag{21}$$

And

$$\begin{aligned} H(y_t) &= H(y_{t-1}) + H(y_t|y_{t-1}) \\ &= H(y_{t-1}) + \frac{1}{2}\log(2\pi e(\sigma^2 + \sigma_{t-1}^2(z_t))), t \in [1,T]. \end{aligned} \tag{22}$$

Then we have

$$\sum_{t=1}^{T} H(y_t) = \sum_{t=1}^{T} (H(y_{t-1}) + \log(2\pi e(\sigma^2 + \sigma_{t-1}^2(z_t)))/2), \tag{23}$$

Algorithm 1 The OLSO Algorithm

Initialization: GP Prior μ_0, σ_0; action space $\mathcal{A}$; the set of historical data $\mathcal{D}_0 = \emptyset$; the weight parameter α.
for $t = 1, 2, \ldots, T$ **do**
  for $a_t \in \mathcal{A}$ **do**
    Observe context c_t;
    Uses posterior inference to predict the mean and the variance for the current action-context pair $z_t = (a_t, c_t)$ based on (19)-(20);
  end
  $a_t = \arg\max_{a_t \in \mathcal{A}} \widetilde{\mu}_t(a_t, c_t) + \beta_t^{1/2} \widetilde{\sigma}_t(a_t, c_t)$;
  $n_t = 0$;
  for *frame* $i = I_t, \ldots, I_{t+1} - 1$ **do**
    if $T_i^{buf} > T_{max}$ **then**
      delete frame i from the buffer;
    else
      Resize frame i to resolution r and send it to server m indicated by a_t;
    end
    if *Received the result of frame* i *and* $L_i < T^{max}$ **then**
      $n_t = n_t + 1$;
    end
  end
  Calculate y_t based on (7);
  Augment data $\mathcal{D}_t = \{\mathcal{D}_{t-1}, (z_t, y_t)\}$;
end

and thus

$$H(y_T) = \sum_{t=1}^{T} \log(2\pi e(\sigma^2 + \sigma_{t-1}^2(z_t)))/2), \tag{24}$$

From Eq.(21) we have

$$\begin{aligned} I(y_T; f) &= \frac{1}{2} \sum_{t=1}^{T} \log(1 + \sigma^{-2} \sigma_{t-1}^2(z_t)) \\ &= \frac{1}{2} \log |\mathbf{I} + \sigma^{-2} \mathbf{K}_T|. \end{aligned} \tag{25}$$

where $\widetilde{\mathbf{K}}_T = [k_z(z, z')]_{z,z' \in Z}$ is the kernel matrix defined over the context-action space.

The corresponding maximum over any set of points $\mathbf{z}_T = (z_1, \ldots, z_T)$ is given by

$$\gamma_T := \max_{\mathbf{z}_T \subset Z : |\mathbf{z}_T| = T} I(\mathbf{y}_T; f). \tag{26}$$

To derive the bound of regret, we first analyze the bound of the maximum mutual information γ_T. For the purpose of analysis, we split the time horizon $\{1, \ldots, T\}$ into smaller blocks of size $\widetilde{N}$ within which f_t does not vary too much. The $\widetilde{N}$ is introduced as a tool in the analysis. Then we have the following lemma.

Lemma 1: The maximum mutual information γ_T in the time-varying setting satisfies

$$\gamma_T \leq (\frac{T}{\widetilde{N}} + 1)(\gamma'_{\widetilde{N}} + \lambda \widetilde{N}^3). \tag{27}$$

where $\gamma'_{\widetilde{N}}$ is the maximum mutual information for the time-invariant setting.

Proof: We assume that $T/\widetilde{N}$ is an integer, and then handle the general case. Considering the fact that the noise sequence ϵ_t is independent, and use the chain rule for mutual information, we have

$$I(\mathbf{y}_T; f) \leq \sum_{i=1}^{T/\widetilde{N}} I(\mathbf{y}_{\widetilde{N}}^{(i)}; f_{\widetilde{N}}^{(i)}), \tag{28}$$

where $\mathbf{y}_{\widetilde{N}}^{(i)} = (y_{\widetilde{N}(i-1)+1}, \ldots, y_{\widetilde{N}_i})$ contains the measurements in the i^{th} block, and $f_{\widetilde{N}}^{(i)}$ is defined analogously. Maximizing both sides over $(z_1, \ldots, z_T)$, we obtain

$$\gamma_T \leq \frac{T}{\widetilde{N}} \gamma_{\widetilde{N}}. \tag{29}$$

To bound $\gamma_{\widetilde{N}}$, we write the relevant covariance matrix as $\widetilde{\mathbf{K}}_{\widetilde{N}} = \mathbf{K}_{\widetilde{N}} \circ \mathbf{H}_{\widetilde{N}} = \mathbf{K}_{\widetilde{N}} + \mathbf{A}_{\widetilde{N}}$, where $\mathbf{A}_{\widetilde{N}} := \mathbf{K}_{\widetilde{N}} \circ (\mathbf{H}_{\widetilde{N}} - \mathbf{1}_{\widetilde{N}})$, and $\mathbf{1}_{\widetilde{N}}$ is a $\widetilde{N} \times \widetilde{N}$ matrix of ones. Since the $(i,j)^{th}$ entry of $\mathbf{H}_{\widetilde{N}} - \mathbf{1}_{\widetilde{N}}$ has absolute value $1 - (1-\lambda)^{\frac{|i-j|}{2}}$, which is upper bounded for all $\lambda \in [0, 1]$ by $\lambda |i - j|$. And each entry of $\mathbf{K}_{\widetilde{N}}$ is between [0,1]. We can obtain the following bound on the Frobenius norm: $||\mathbf{A}_{\widetilde{N}}||_F^2 \leq \sum_{i,j} (i-j)^2 \lambda^2 \leq \widetilde{N}^4 \lambda^2$. Using the above inequality, we bound γ_N via Mirsky's theorem [30], [31], which is given as follows.

$$\begin{aligned} \gamma_{\widetilde{N}} &\leq \gamma'_{\widetilde{N}} + \widetilde{N} \log(1 + \lambda \widetilde{N}^2) \\ &\leq \gamma'_{\widetilde{N}} + \lambda \widetilde{N}^3, \end{aligned} \tag{30}$$

where $\gamma'_{\widetilde{N}}$ is the maximum mutual information for the time-invariant setting. Combining (29) and (30), we can obtain the bound for γ_T. Since we consider the case that $T/\widetilde{N}$ is an integer, we hence add one to $T/\widetilde{N}$ for general cases. Then we can get (27). □

Based on the above definitions and lemma, similar to [32], we can have the following theorem which provides regret bounds for our proposed algorithm.

Theorem 1: Suppose the domain $Z \subseteq [0, c]^d$ is compact and convex, where $d \in \mathbb{N}$, $c > 0$. And suppose that the kernel $k_z(z, z')$ satisfies the following condition for some constants $a, b > 0$ and all $L \geq 0$:

$$\mathbf{Pr}(sup_{z \in Z} |\frac{\partial f(z)}{\partial z_j}| > L) < a e^{(-L/b)^2}, \quad j = 1, \ldots, d \tag{31}$$

and set $C_1 = 8/\log(1 + \sigma^{-2})$, $\beta_T = 2\log(\pi^2 T^2)/2\delta + 2d \log(cdbT^2 \sqrt{\log(da\pi^2 T^2)/2\delta})$, the algorithm satisfies the following:

$$R_T \leq \sqrt{C_1 T \beta_T \gamma_T} + 2 \tag{32}$$

with probability at least $1 - \delta$, where $\widetilde{N} \in \{1, \ldots, T\}$.

Proof: To measure the bound of the distance between two points (action-context pairs) in the domain Z, we first discretize the domain Z with size $(h_t)^d$ satisfying

$$||z - [z]_t|| \leq cd/h_t, \quad \forall z \in Z, \tag{33}$$

where $[z]_t$ denotes the closest point in Z to z.

Then, based on the results in [32], by analyzing three high probability events regarding β_t, we have $|f_t(z) - f_t([z]_t)| \leq$

$L_t cd/h_t$, where $L_t = b\sqrt{\log(da\pi^2 t^2/2\delta)}$. By setting $h_t = cdt^2 L_t$, we have

$$|f_t(z) - f_t([z]_t)| \leq \frac{1}{t^2} \tag{34}$$

With the triangle inequality applied, the maximizing point z_t^* at time t is found to satisfy $|f_t(z_t^*) - \widetilde{\mu}_{t-1}([z_t^*]_t)| \leq \beta_t^{1/2}\widetilde{\sigma}_{t-1}([z_t^*]_t) + 1/t^2$. Then the instantaneous regret is bounded as follows:

$$r_t = f_t(z_t^*) - f_t(z_t) \leq 2\beta_t^{1/2}\widetilde{\sigma}_{t-1}(z_t) + 1/t^2. \tag{35}$$

Finally, the cumulative regret is bounded as:

$$\begin{aligned} R_T &= \sum_{t=1}^{T} r_t \\ &\leq \sum_{t=1}^{T}(2\beta_t^{1/2}\widetilde{\sigma}_{t-1}(z_t) + 1/t^2) \\ &\leq \sqrt{T\sum_{t=1}^{T} 4\beta_t\widetilde{\sigma}_{t-1}^2(z_t)} + 2 \\ &\leq \sqrt{C_1 T\beta_T\gamma_T} + 2. \end{aligned} \tag{36}$$

□

For the squared exponential kernel, from [25] we have $\gamma'_{\widetilde{N}} = \mathcal{O}(\log \widetilde{N}) = \mathcal{O}^*(1)$. The $\mathcal{O}^*$ notation is a variant of $\mathcal{O}$, where log factors are suppressed. And thus we obtain

$$(\frac{T}{\widetilde{N}} + 1)(\gamma'_{\widetilde{N}} + \lambda\widetilde{N}^3) = \mathcal{O}^*((\frac{T}{\widetilde{N}} + 1)(1 + \lambda\widetilde{N}^3)). \tag{37}$$

Setting $\widetilde{N} = \mathcal{I}(\lambda^{-1/3})$, where $\mathcal{I}(x)$ denotes the integer which is closest to $x \in \mathbb{R}$. We find that γ_T behaves as $\mathcal{O}^*(\lambda^{1/3}T)$ when $\lambda \geq 1/T^3$, and as $\mathcal{O}^*(1)$ when $\lambda < 1/T^3$. Substitute these into Theorem 1, we obtain that the regret is bounded by $\mathcal{O}^*(\max\{\sqrt{T}, T\lambda^{1/6}\})$, which means our algorithm can achieve a sub-linear deviation of the optimal solution, i.e., the regrets increases sub-linearly as time increases.

V. Performance Evaluations

In this section, we evaluate the performance of the proposed algorithm (OLSO) and compare it with other algorithms.

A. Evaluation Setup

In the evaluation, the video frame resolution is set to 220×220, 280×280, 340×340, 400×400, and 460×460 pixels respectively. We use a subset of videos from the FCVID dataset [33] to train ResNet-50 CNN models for activity recognition. The average recognition accuracy with respect to different resolutions is profiled and shown in Fig. 4. We profile the computation demand of CNN models with different image resolutions by running these models on a desktop, which includes the processing time and the corresponding CPU rate. Then the computation demand is estimated as the average required CPU cycles per video frame. Here, we measure the computation demand by CPU cycles since the current server resource trace is based on CPU. Note that our offloading algorithm can automatically adapt based on the available server resource (GPU or CPU). If GPU is used at the server, the frame processing rate will be higher and then our algorithm may offload more frames to the server. However, our algorithm does not change, and the only change is the input parameter which is similar to changing other parameters related to the server workload.

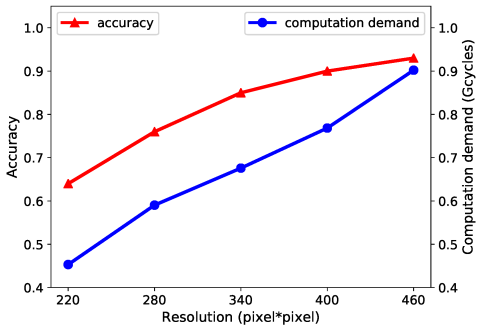


Fig. 4. Accuracy and computation cost vs. resolution.

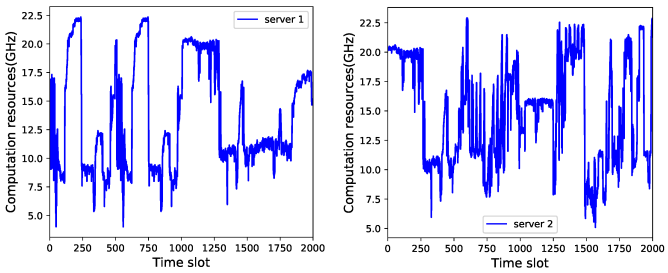


Fig. 5. Examples of server resource traces.

In the evaluation, there are three servers, and their available CPU resources change with time. To emulate the dynamic server resources status, we use a data trace collected based on distributed datacenters from Bitbrains [34], which records the CPU resources status of 1750 VMs. We randomly select the data traces of several VMs and use their CPU resources status as those of our servers. Fig. 5 shows some sample traces of the server computation resources. To emulate the dynamic bandwidth environment, we use the bandwidth traces from the dataset HSDPA [35]. Fig. 6 shows an example bandwidth trace. Video frames processed by the local CNN model are scaled to 220×220, with the mean processing time of 180ms per frame. The time constraint for each frame is set to be 200 ms. In our experiment, the length of one time slot is set to be one second. The GP prior μ_0 and σ_0 are set to be 0 and 0.5 respectively, and the length-scale l is set to be 1. The running time of the OLSO algorithm is less than 10 ms and it is negligible compared to the length of one time slot.

We compare our algorithm with the following algorithms:

- *Local*: All frames are processed locally using the local CNN model with the lowest resolution.
- *Accuracy-opt*: This algorithm selects the highest resolution for all frames to maximize accuracy. It first explores all servers and then sticks to the server with the minimum delay.

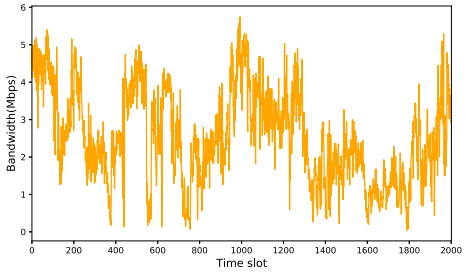
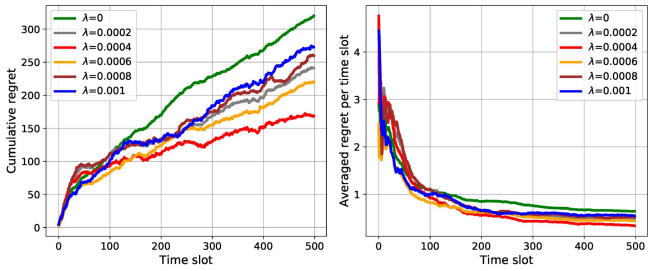
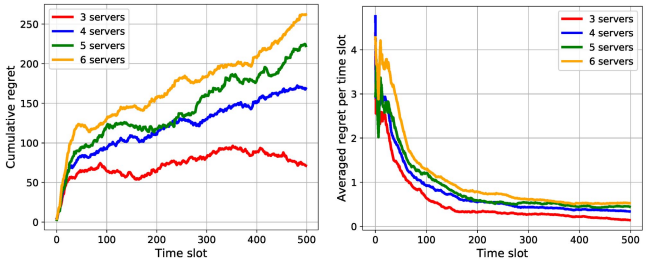


Fig. 6. An example of bandwidth traces.



(a) Cumulative regret (b) Time-averaged regret

Fig. 7. Regret v.s. λ.



(a) Cumulative regret (b) Time-averaged regret

Fig. 8. Regret vs. number of servers.

- *LinUCB*: This algorithm selects the server and resolution based on the minimum upper confidence bound of the utility according to the classic MAB theory [23].
- *Optimal*: This algorithm assumes a priori knowledge about the server resources, network bandwidth, etc. It calculates the utility of all possible combinations of server and resolution, and then chooses the server and resolution that maximize the utility. This algorithm is impossible in practice, so it only provides a performance upper bound.

We use videos in the FCVID dataset for video analytics and analyze their accuracy, processing rate and utility under the above algorithms.

B. Regret Analysis and Performance Ratio

We first evaluate the regret of our algorithm (OLSO) and analyze how the key parameter λ and the number of servers influence the regret. To evaluate the influences of λ, we set the number of servers to be four, and $\alpha = 8$. The network latency is randomly and uniformly distributed between 20 ms and 50 ms. For simplicity, we compare the cumulative regret and average regret per time slot with different values of λ.

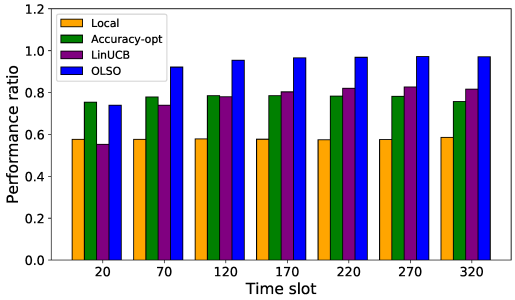
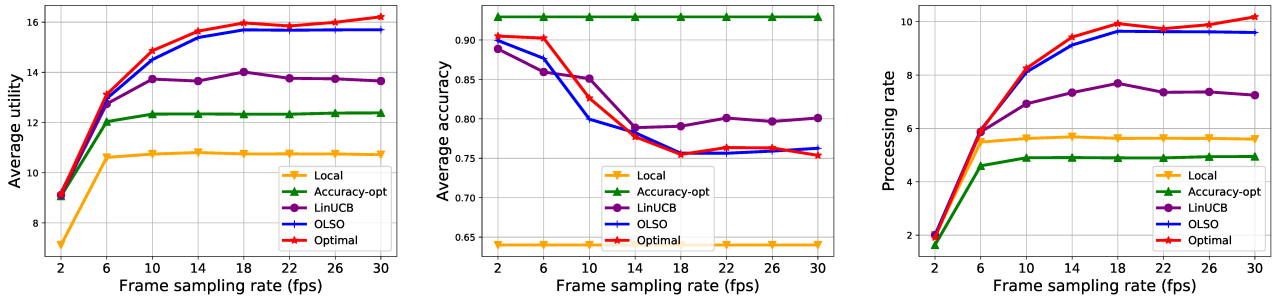


Fig. 9. The performance ratio relative to the optimal algorithm after a certain number of time slots.

We tune the parameter λ and find that when $\lambda \geq 0.001$, the performance decreases significantly. Then we find the best value of λ by grid search between 0 and 0.001. Fig. 7 shows the convergence of OLSO with λ to be 0, 0.0002, 0.0004, 0.0006, 0.0008, and 0.001. In general, a smaller λ puts more weight on the historical data, and a larger λ emphasizes more on the recent data. When $\lambda = 0.0004$, the algorithm achieves the lowest regret after 100 time slots, and it achieves a sub-linear increasing trend. Fig. 7(b) shows the average regret per time slot of our algorithm as time goes. The first several time slots have high regret because of exploration. After 100 time slots, the average regret per time slot reaches a low level of less than 1. In the following experiments, we set $\lambda = 0.0004$.

In Fig. 8, we compare the cumulative regret and the time-averaged regret under different number of available servers. The number of servers affects the size of the action space, and thus affects the performance of the OLSO algorithm. From Fig. 8(a) we observe that when the number of servers becomes larger, the cumulative regret in the same time period also becomes higher. This is because when more servers are available, it requires more exploration time, which leads to longer convergence time and thus higher cumulative regret. However, the OLSO algorithm always achieves a sub-linear increasing trend under different number of servers. From Fig. 8(b), we can also observe that the time-averaged regret of all cases can reach a similar low level of less than 1, which verifies that our algorithm can finally achieve similar performances with the optimal solution under varying number of available servers. In the following experiments, we set the number of servers to be four.

In Fig. 9, we compare OLSO with other algorithms in terms of performance ratio, which is the relative achieved utility compared to that of the Optimal. As can be seen, our algorithm has the highest performance ratio and it approaches the optimal as the number of time slots above 120. At the beginning, when the number of time slots (e.g., 20) is few, the performance ratio of our algorithm is low. This is because it does not have prior knowledge about the server and the network condition, and can only randomly select server and resolution to gather information. Through exploration, our algorithm acquires more knowledge about the system and achieve better performance as time goes, and gradually approaches to the performance of the Optimal algorithm.



(a) Utility (b) Accuracy (c) Processing rate

Fig. 10. The impact of frame sampling rates.

LinUCB also learns more knowledge as time goes and performs better as time goes, but it still underperforms our algorithm. This is because LinUCB assumes a linear relationship between context and reward, which may not be appropriate and then results in inaccurate learning results and wrong decisions. Besides, our algorithm can better adapt to environment updates, and thus can better learn the server status and make a better decision. Compared with other algorithms, the Accuracy-opt algorithm can achieve a higher performance ratio at the beginning, but becomes worse as time goes. This is because it only explores each server once and then sticks to the server with the minimum delay at that time, and thus achieves better performance at the beginning. However, the server resources change over time, and then its performance may become worse over time. Moreover, the Accuracy-opt algorithm does not select frame resolution dynamically based on the server resources, and thus does not have a high performance ratio. The local algorithm generally has the worst performance because it always selects the lowest resolution and processes frames locally.

C. The Impact of Frame Sampling Rate

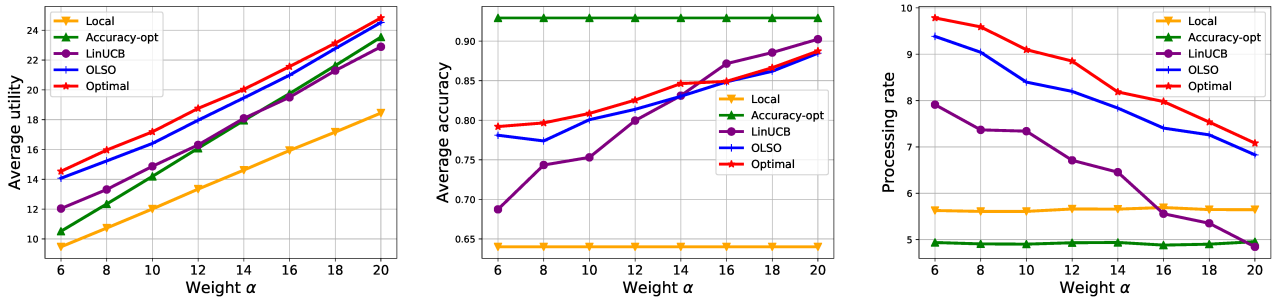
Fig. 10 shows the impacts of the frame sampling rate on different algorithms. In the evaluation, we set $\alpha = 8$. The network latency is randomly and uniformly distributed between 20 ms and 50 ms. As shown in Fig. 10(a), the OLSO algorithm outperforms Local, Accuracy-opt and LinUCB algorithms, and has similar utility to the Optimal algorithm. Fig. 10(b) shows the corresponding recognition accuracy which is determined by the frame resolution, and thus it can partially reveal their resolution selection strategy when the frame sampling rates change. Fig. 10(c) shows the corresponding frame processing rate achieved by these algorithms. When the sampling rate is low (e.g., 2 fps), the frame processing rate is limited by the sampling rate since there are not always frames in the buffer to be processed, and then the utility is mainly affected by accuracy. Thus, all algorithms except the Local algorithm have similar performance as shown in Fig. 10(a), because all of them tend to select higher resolution and thus they achieve similar accuracy as shown in Fig. 10(b). The Local algorithm has the lowest utility because it selects the lowest resolution which has the lowest accuracy.

When the frame sampling rate increases, the frame processing rate also increases and thus the utility of all algorithms improves. When the frame sampling rate is high (e.g., higher than 18 fps), the buffer is always full and the processing rate is restricted by the transmission and processing time of the video frames, and thus the utility of all methods is not affected much by the frame sampling rate. In this case, selecting the highest resolution does not achieve the best utility since it limits the frame processing rate. LinUCB and OLSO are able to adjust the resolution that can achieve better tradeoff between accuracy and processing rate. Besides, LinUCB and OLSO dynamically select the server with more resources and achieve higher frame processing rate as shown in Fig. 10(c). Thus they can achieve better utility than Accuracy-opt. OLSO has better learning ability and thus outperforms the LinUCB algorithm. OLSO improves the utility by up to 14.3% compared to the LinUCB algorithm.

D. The Impact of α

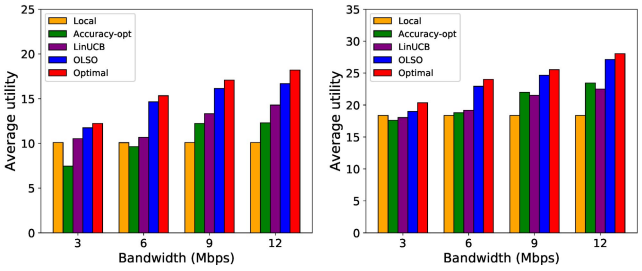
Fig. 11 shows the impact of the weight parameter α on the performances of different algorithms. We set the frame sampling rate to be 15 fps, and the network latency is uniformly distributed between 20 ms and 50 ms. As shown in Fig. 11(a), OLSO has the best performance if the Optimal algorithm is not considered, and the Local algorithm has the worst performance. LinUCB outperforms Accuracy-opt when α is small, but underperforms Accuracy-opt when α is large. When α is small, the frame processing rate has more weight in calculating the utility. The Accuracy-opt algorithm selects the highest resolution which has the longest processing time and transmission time, resulting in a low processing rate and hence lower utility. When α is large, the accuracy has more weight in calculating the utility. The Accuracy-opt algorithm selects the highest resolution, resulting in higher utility, but it is still lower than our OLSO algorithm.

Fig. 11(b) and Fig. 11(c) present the corresponding accuracy and frame processing rate achieved by these algorithms under different α. The Local and Accuracy-opt algorithms select fixed resolutions and thus their accuracy and processing rate do not change with α. As α increases, the accuracy of LinUCB, OLSO and Optimal also increases, because accuracy has more weight in calculating utility and these algorithms tend to select higher resolutions so that a better utility can be achieved. Our OLSO algorithm has similar performance with the Optimal algorithm under different α, which validates that our algorithm can achieve the best tradeoff between accuracy and processing rate.



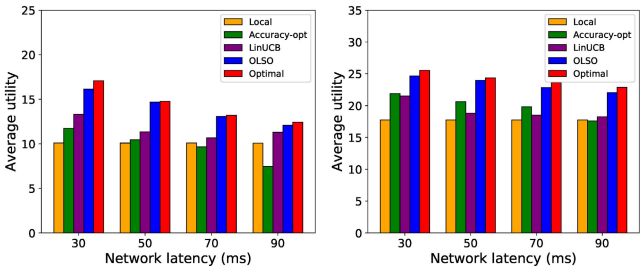
(a) Utility (b) Accuracy (c) Processing rate

Fig. 11. The impact of the weight parameter α.



(a) $\alpha = 6$ (b) $\alpha = 20$

Fig. 12. Average utility of different approaches under different network bandwidth.



(a) $\alpha = 6$ (b) $\alpha = 20$

Fig. 13. Average utility of different approaches under different network latency.

E. The Impact of Network Condition

The network condition affects the communication delay, and thus affects the performance of the offloading algorithms. Fig. 12 compares the performance of the five algorithms under various network bandwidth. We set the network latency to be 30 ms, the frame sampling rate to be 15 fps, and the weight parameter α to be 6 and 20 respectively. As shown in the figure, OLSO achieves higher utility compared with Local, Accuracy-opt and LinUCB under all cases. Specifically, OLSO improves the utility by up to 70.0% compared to the Local algorithm, and by up to 22.3% compared to the best baseline algorithm (i.e., LinUCB algorithm). The performance of the Local algorithm remains the same under different network bandwidth because it does not upload frames, and thus its utility is not affected. The utility of the other four algorithms increases as the bandwidth increases. This is because higher bandwidth results in lower transmission delay, and thus more frames can be processed within the same time, improving processing rate and then utility.

Compare Fig. 12(a) and Fig. 12(b), we find that when the bandwidth is high, the Accuracy-opt algorithm has better performance under a larger α. This is because when α is large, accuracy has more weight in calculating the utility and thus the Accuracy-opt algorithm achieves higher utility. However, when the bandwidth is low, the utility of Accuracy-opt drops dramatically under small α. This is because Accuracy-opt always selects the highest resolution which takes more time to upload frames. When the bandwidth is low, more frames cannot be processed within the time constraint, resulting in lower processing rate.

Fig. 13 compares the performance of different algorithms under different network latency. We set the network bandwidth to be 8 Mbps, the frame sampling rate to be 15 fps, and also evaluate the performances under both small and large α. The OLSO algorithm achieves the highest utility except the Optimal algorithm under various network latency. Similar to the reasons explained above, the performances of all the algorithms except the Local algorithm decrease as the network latency increases. When the network latency increases, fewer frames can be processed within the same time, thus lowering the frame processing rate and then utility of these four algorithms. Among all algorithms, Accuracy-opt is affected most by the network latency. This is because Accuracy-opt always selects the highest frame resolution which takes more time to process, and then more frames cannot be processed within the time constraint when the network latency is high, hence decreasing its utility.

VI. Conclusion

In this paper, we studied the server resource-aware offloading problem for video analytics, where the goal is to maximize the utility which is a weighted function of accuracy and frame processing rate. To overcome the challenge of unknown server resources and network condition, we formulated the problem as a contextual MAB problem and proposed an efficient online learning based offloading algorithm to solve it. The proposed algorithm selects server and resolution based on user's local observations, e.g., historical delay and accuracy. To make the algorithm adaptable for time-varying environments, we assign different weights to samples collected at different time, e.g., higher weights to more recent samples. Through extensive evaluations, we demonstrated the effectiveness of our proposed algorithm. Specifically, our proposed algorithm improves the utility by up to 70.0% compared to running video analytics locally, and by up to 22.3% compared to the best offloading algorithm (i.e., LinUCB algorithm).

As future work, we will conduct more thorough experiments by considering other type of server resources (e.g., GPU) and local resources (e.g., NPU [36]). Besides, other video configurations (e.g., frame sampling rate) may affect the quality of video analytics, and thus should be considered in our utility formulation. In addition to video analytics, the proposed algorithm can also be applied to server selection in other computational extensive applications such as augmented/virtual reality.

REFERENCES

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1–9.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[3] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 756–764.

[4] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. ACM MobicCom*, Aug. 2019, pp. 1–16.

[5] X. Ge, S. Tu, G. Mao, C.-X. Wang, and T. Han, "5G ultra-dense cellular networks," *IEEE Wireless Commun.*, vol. 23, no. 1, pp. 72–79, Feb. 2016.

[6] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 10–18.

[7] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 377–392.

[8] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 253–266.

[9] T. Lu, D. Pál, and M. Pál, "Contextual multi-armed bandits," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 485–492.

[10] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "AWStream: Adaptive wide-area streaming analytics," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 236–252.

[11] T. Tan and G. Cao, "FastVA: Deep learning video analytics through edge processing and NPU in mobile," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 1947–1956.

[12] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 257–266.

[13] T. Tan and G. Cao, "Deep learning video analytics through edge computing and neural processing units on mobile devices," *IEEE Trans. Mobile Comput.*, early access, Aug. 19, 2021, doi: 10.1109/TMC.2021.3105953.

[14] T. Tan and G. Cao, "Deep learning video analytics on edge computing devices," in *Proc. IEEE SECON*, Jul. 2021, pp. 1–9.

[15] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[16] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[17] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 1279–1287.

[18] Y. Liu, H. Xu, and W. C. Lau, "Online job scheduling with resource packing on a cluster of heterogeneous servers," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 1441–1449.

[19] H. Liu and G. Cao, "Deep reinforcement learning-based server selection for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13351–13363, Dec. 2021.

[20] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE INFOCOM*, Apr. 2019, pp. 1468–1476.

[21] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep residual learning for image compression," in *Proc. CVPR Workshops*, 2019, pp. 1–5.

[22] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, nos. 2–3, pp. 235–256, 2002.

[23] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 661–670.

[24] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1–9.

[25] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger, "Information-theoretic regret bounds for Gaussian process optimization in the bandit setting," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 3250–3265, May 2012.

[26] C. Y. Deng, "A generalization of the Sherman–Morrison–Woodbury formula," *Appl. Math. Lett.*, vol. 24, no. 9, pp. 1561–1564, Sep. 2011.

[27] R. Zhu, Q.-G. Fei, D. Jiang, and Z.-F. Cao, "Dynamic sensitivity analysis based on Sherman–Morrison–Woodbury formula," *AIAA J.*, vol. 57, no. 11, pp. 4992–5001, Nov. 2019.

[28] V. Van Vaerenberg, M. Lázaro-Gredilla, and I. Santamaría, "Kernel recursive least-squares tracker for time-varying regression," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 8, pp. 1313–1326, Aug. 2012.

[29] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2012.

[30] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[31] L. Mirsky, "Inequalities and existence theorems in the theory of matrices," *J. Math. Anal. Appl.*, vol. 9, no. 1, pp. 99–118, Aug. 1964.

[32] I. Bogunovic, J. Scarlett, and V. Cevher, "Time-varying Gaussian process bandit optimization," in *Proc. Artif. Intell. Statist.*, 2016, pp. 314–323.

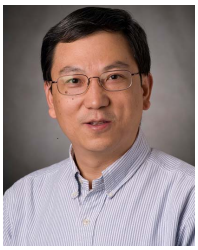
[33] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang, "Exploiting feature and class relationships in video categorization with regularized deep neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 352–364, Feb. 2017.

[34] S. Shen, V. Van Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 465–474.

[35] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. MMSys*, 2013, pp. 114–118.

[36] T. Tan and G. Cao, "Efficient execution of deep neural networks on mobile devices with NPU," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2021, pp. 283–298.

Heting Liu received the B.S. degree in electrical engineering from Sun Yat-sen University, China, in 2016. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Pennsylvania State University, USA. Her research interests include edge computing, mobile cloud computing, wireless networks, and machine learning.



Guohong Cao (Fellow, IEEE) received the B.S. degree in computer science from Xi'an Jiaotong University and the Ph.D. degree in computer science from The Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering, Pennsylvania State University, where he is currently a Distinguished Professor. He has published more than 200 papers in the areas of wireless networks, mobile computing, machine learning, wireless security and privacy, and the Internet of Things, which have been cited over 20000 times. He is a fellow of the AAAS. He has received several best paper awards, the IEEE INFOCOM Test of Time Award and the NSF CAREER Award. He has served on the editorial board of IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Co-Chair of IEEE SRDS, MASS, and INFOCOM.