AutoML for Video Analytics with Edge Computing

Apostolos Galanopoulos*, Jose A. Ayala-Romero*, Douglas J. Leith*, George Iosifidis†

* School of Computer Science and Statistics, Trinity College Dublin, Ireland
† Delft University of Technology, The Netherlands

***Abstract*—Video analytics constitute a core component of many wireless services that require processing of voluminous data streams emanating from handheld devices. Multi-Access Edge Computing (MEC) is a promising solution for supporting such resource-hungry services, but there is a plethora of configuration parameters affecting their performance in an unknown and possibly time-varying fashion. To overcome this obstacle, we propose an Automated Machine Learning (AutoML) framework for jointly configuring the service and wireless network parameters, towards maximizing the analytics' accuracy subject to minimum frame rate constraints. Our experiments with a bespoke prototype reveal the volatile and system/data-dependent performance of the service, and motivate the development of a Bayesian online learning algorithm which optimizes on-the-fly the service performance. We prove that our solution is guaranteed to find a near-optimal configuration using *safe* exploration, i.e., without ever violating the set frame rate thresholds. We use our testbed to further evaluate this AutoML framework in a variety of scenarios, using real datasets.**

***Index Terms*—Edge Computing, Automated Machine Learning, GP-UCB**

I. Introduction

Background & Motivation. An increasing number of wireless services today rely on accurate and fast video analytics. From mobile gaming and augmented reality apps [1], to cognitive assistance [2], autonomous vehicles or surveillance systems [3], user devices capture frames (or, images) from video streams that need to be processed so as to extract critical information, e.g., detect objects of interest. These video analytics are very demanding as they require voluminous data transfers and daunting computations, in almost real time. This renders impractical both their execution at the resource-constrained user devices, and their transfer to distant cloud servers [4]. A promising solution for adhering to these requirements is, indeed, Multi-Access Edge Computing (MEC) [5], where users transmit their video frames for processing to nearby servers equipped with GPUs that implement efficient Neural Networks (NNs).

Existing MEC solutions manage computing or network resources to offload various tasks from user devices [6]–[8]. However, video analytics are heavily affected by several *new* parameters at the user side, e.g., image resolution and encoding rate, and at servers, e.g., NN architecture. In particular, the key criteria of accuracy and latency are intertwined and shaped by the configuration parameters of the processing pipeline[1] and the wireless network that connects devices and servers. For instance, sending low-resolution or compressed frames reduces the transmission latency but also the object recognition accuracy; and increasing the frame rate improves a user's experience but strains the network and exacerbates the frame rate of others. Clearly, deciding jointly the resource allocation and pipeline configuration is of utmost importance.

[1]This term refers to the video processing stages, e.g., decoder, frame sampler and inference module (as, e.g., Yolo [9]), see [10] for details.

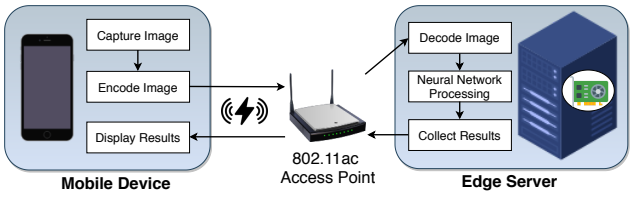


Fig. 1: Video edge analytics prototype where mobile devices employ Neural Network processing units at edge servers to execute high-accuracy, low-latency object recognition in their frames.

Pertinent studies focus on reducing the resource costs of video analytics [10], [11], and on maximizing their performance [12], [13]; see Sec. II for an overview. However, the dependence of performance metrics – accuracy, latency, and others – on the pipeline configuration and on the allocated resources is unknown in practice, and might as well vary with time (e.g., due to wireless conditions). Importantly, as our experiments reveal, the performance depends also on the *platform*, i.e., the devices' and servers' hardware and software; and on the actual video data. Hence prior approaches that rely on statistical models, offline datasets or pre-training, are limited to specific systems and scenarios. Here we take a fundamentally different approach and develop a *Bayesian learning framework towards automating the configuration of multi-user video edge analytic services*.

Methodology & Contributions. We start with an experimental analysis using our prototype system where devices capture video frames and transmit them to an edge server that performs object recognition using a Deep Neural Network (DNN), Fig. 1. We aim to maximize the recognition accuracy while satisfying user-defined minimum frame rate constraints; by deciding the image encoding rate, service time allocation and NN input layer size. We find that the system has stochastic accuracy and latency response (which shapes the achieved frame rate) even for fixed configurations. The former is due to differences in the images' objects, and the latter due to wireless channels and processing delay variations. We also find that similar configurations induce similar performance, which depends on the DNNs and devices, and exhibits even non-monotonic behavior. Our measurements extend prior works [13]–[15], and highlight the platform *and* data-dependent performance of these systems.

Motivated by these findings, we propose an optimization framework consisting of two components: a surrogate model builder for the unknown objective and constraint functions, and an acquisition rule that explores iteratively the system configurations. The former employs Gaussian Processes (GP) and Bayesian updates [16] to construct the required models in real time using the collected data. The second component quantifies each configuration's optimality, while also accounting for the uncertainty regarding the existence of better configurations [17]. The result is a data-driven, platform-oblivious algorithm that is executed at system runtime. We prove that the algorithm finds a near-optimal solution and achieves average *sublinear pseudo regret* of $O(\sqrt{T}\gamma_T)$ where γ_T is a system-related parameter. Moreover, the algorithm is *safe* in the sense that it satisfies the users' minimum frame rate constraints while exploring the configuration space.

Our approach builds on the theory of Bayesian non-parametric learning, and falls into the realm of Automated Machine Learning. *AutoML*, as it is known, has been used to automate the configuration of software packages[2], or the selection of various ML hyper-parameters [18], [19] which otherwise are set using heuristics [20]. We extend these ideas to automate the video pipeline and network configuration, while catering for frame rate constraints. This way, we tackle the main challenge of the service's dependency on system hardware and video data. Being a powerful framework, it can be used to also allocate computing resources, select different networks, and so on (see details in Sec. VI).

Finally, we evaluate the system performance and find that our algorithm can get to within 5% from the optimal point in no more than 150 iterations. We also propose a set of practical steps to improve its performance, based on our experimental observations, e.g., the usage of stopping criteria for the different stages of the algorithm. Our technical contributions can be thus summarized as follows:

• Experimentally-motivated problem. We perform extensive experiments using different system equipment and datasets which reveal the volatile performance of video analytics and their dependency on said system and data.

• AutoML Framework. We propose an optimization framework that finds a near-optimal configuration without violating the users' frame rate thresholds. This is achieved by combining a Bayesian GP technique with bandit learning and safe constraint exploration. To the best of our knowledge, this is the first time an AutoML framework is used to configure a video edge analytics service.

• Model Extensions. We extend our analysis to problems where additional video-related (e.g., frame resolution) or network parameters (e.g., user association to networks/servers) are decided. This manifests the framework's potential.

• Prototypes and Experiments. We evaluate the framework based on real data in our bespoke prototype, where we perform a thorough parameter sensitivity analysis, quantify its overheads, and verify its generality using a wealth of scenarios and system setups. All our measurements are made available in an online fully-documented dataset [21].

[2]E.g., the many parameters of mathematical solvers such as IBM CPLEX.

Paper Organization. Sec. II discusses prior work, Sec. III presents our preliminary experiments, and Sec. IV introduces the system model and problem. Sec. V presents the AutoML framework, while in Sec. VI we discuss interesting extensions and practical aspects of our approach. Finally, Sec. VII presents the evaluation, and Sec. VIII our conclusions.

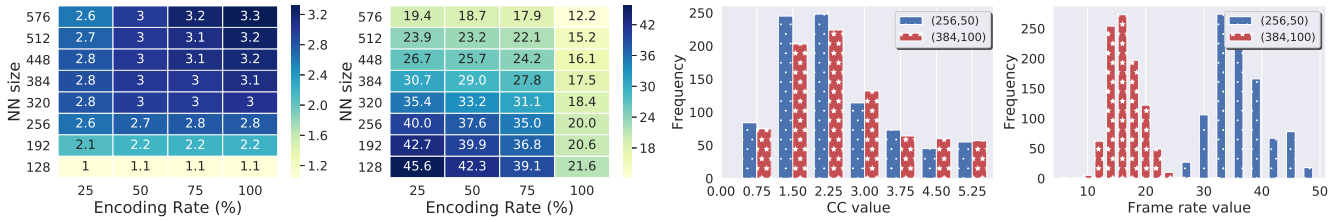
II. Related Work

Systems & Experiments. Video analytics often employ MEC to improve scalability and latency. For instance, [22] and [23] explore DNN partitioning across user devices, edge servers and the cloud. Other systems like JAGUAR [24], Glimpse [25], OverLay [26] and [27] improve real-time object detection via intelligent encoding, caching, visual space pruning, and on-device tracking, respectively. Additionally, JALAD [28], MobiQoR [29] and DeepDecision [13] explore the accuracy - latency trade off. These important experimental studies reveal the gains of proper configurations, while our experiments further explore the configuration dependence on the system architecture and video data.

Optimization-based Approaches. Configuring these systems, however, requires solving rigorous mathematical problems. DeepDecision [13] formulates a frame-rate/accuracy problem to decide the sampling and DNN model; VideoStorm [14] allocates resources through a utility maximization problem with greedy search of configurations; and [29] minimizes the energy and latency. A convex program is solved in [30] to decide frame sizes, and [31] formulates a similar integer decision problem; while [32] minimizes latency. Another large corpus of works take a computation-offloading perspective and handle computing or network resources without considering video metrics (e.g., accuracy) or NN configurations (e.g. NN size); see [6]–[8] and survey [33]. Importantly, all above works formulate *static* model-driven problems, while in practice training data are hardly available and models are valid only for specific systems.

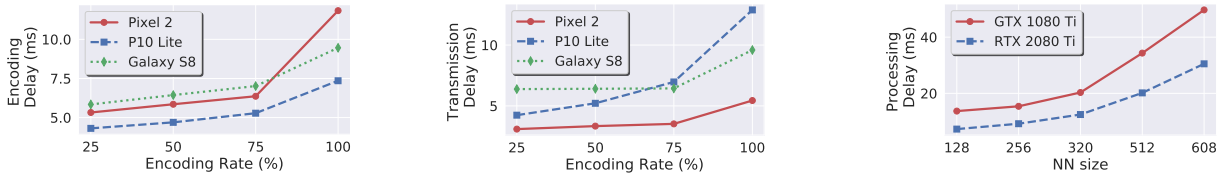
Learning & Adaptive Approaches. Dynamic algorithms can indeed tune better such systems. Chameleon [10] profiles periodically the configurations and searches greedily for the most resource-prudent, but does not consider latency; and VideoEdge [11] solves a similar problem for hierarchical systems. An integer program is solved in [34] to allocate computing resources and decide the image compression and DNN model. In [12], video quality and computing resources are selected to maximize successful queries; while [35] uses a Lyapunov algorithm to select frame and NN parameters, and optimizes accuracy under *average* delay constraints. These interesting works either do not offer optimality bounds [10], [34] or consider asymptotic-only performance [35], assume known models [12], [34], [35] or convex functions [12]. Finally, online algorithms for *general* edge computing, e.g., [36], [37], do not cater for video analytic metrics or the specifics of video pipelines.

Our approach is different since: *(i)* it uses Gaussian Processes [16] to build models in real-time, thus does not



(a) Cumulative Confidence (b) Frame Rate (c) CC distribution (d) Frame Rate distribution

Fig. 2: (a)-(b): Cumulative Confidence and frame rate for various NN sizes and encoding rates; results are averaged across $32K$ images of COCO dataset [38]. (c)-(d): Distributions of CC and frame rate for (NN size, encod. rate): $(256, 50\%)$, $(384, 100\%)$.



(a) (b) (c)

Fig. 3: (a) Encoding and (b) transmission delay for 3 devices. (c) Dependence of the NN processing delay (YOLO [9]) on GPU.

require prior data; *(ii)* jointly configures server, device and network parameters; and *(iii)* employs data-efficient non-parametric learning, hence does not make assumptions about the system. We draw ideas from the area of Automated Machine Learning (AutoML) that streamlines the selection of ML hyper-parameters cf. [18], [19], [39], also using, lately, Bayesian optimization to improve the overall process [20]. Such techniques have been only recently used in systems, e.g., configuring cloud servers [40] or cellular networks [41].

III. Preliminary Experiments

In this section, we lay out our testbed details, and present measurements that motivate our optimization approach.

Testbed setup. We built a bespoke Android application that captures images though the mobile's camera, performs JPEG *encoding*, and transmits the compressed images[3] to a MEC server through a wireless 802.11ac Access Point (AP)[4], Fig. 1. A C/C++ routine at the server uses the REST API for accepting object recognition requests on the transmitted frames. It firstly decodes the JPEG files to obtain RGB images, and then downsamples them to match the input layer size of the DNN at the server's GPU. The integer RGB image values are converted to floats before processed with the state-of-the-art object recognition system YOLO [9], that accepts an $y \times y$ array of image pixels. Henceforth, the dimension y is referred to as the NN input layer size, or simply the *NN size*. The output is a set of: *(i)* bounding box coordinates, *(ii)* inferred classes, and *(iii)* confidence values for each recognized object. Those are transmitted back to the devices and overlaid on their screens. The main configurable parameters of this system are the image encoding rate x, which determines frame quality and file size, and the NN size y that affects the inference quality and delay.

Results. Previous works, e.g., [13]–[15], have studied similar trade-offs between such system knobs in an offline setup, i.e., by pre-calculating the average Precision/Recall accuracy for large datasets of images. However, we aim to automate the system configuration at runtime, and hence cannot rely on offline evaluations; instead, we need *instantaneous* feedback for the performance. To that end, we use the Cumulative Confidence (CC) which is simply the sum of confidence values for all recognized objects that is outputted by YOLO, and is instantly available for each processed frame. We first quantify the trade-off between the CC and service frame rate using the COCO dataset of 40K images. Figures 2a-2b depict the average CC and achieved frame rate, for different encoding rates and NN input sizes. It is evident that increasing the NN size and/or encoding rate, increases the CC and decreases the frame rate. Interestingly, we also found in Fig. 2a a case of non-increasing impact of the NN size on CC (for 25% encoding rate). Notice that the CC increases with the NN size before dropping for NN size > 448.

The main issue with those results is that they are averages of the system performance and can be obtained only after applying object recognition to thousands of images for each possible system configuration. Indeed, Figures 2c-2d depict the variations in CC and frame rate for 2 specific configurations. Moreover, the observed increase (decrease) in CC (frame rate) is non-linear with either NN size or encoding rate, and surprisingly, not even monotonic as explained above. The measured performance can also vary depending on factors like the device environment and specifications, channel conditions or server capabilities, making the development of general accuracy and latency models highly cumbersome. We demonstrate the above in Fig. 3a-3b, where we measure the average encoding and transmission delay

[3]Encoding an image at a certain rate, e.g. 50% achieves 2 things. First, the image data is converted to the JPEG format, and second the resulted file is compressed to 50% of the original file size, hampering image quality.

[4]The AP is the ASUS RT-AC86U router, and the server a 3.7 GHz Core i7, 32 GB RAM PC, with a GeForce RTX 2080 Ti GPU.

respectively, for 3 different mobile devices. Clearly, although all delays are increasing with the encoding rate, the fitted curves vary substantially across devices. The same trend persists when the server's hardware (GPU) changes. Fig. 3c depicts the difference in DNN processing delay between 2 GPUs as we increase the NN input size.

In summary: our experiments reveal a non-trivial multi-modal impact of the encoding rate and NN size on CC and frame rate. These 2 key metrics are platform and dataset dependent, highly volatile, and there are unknown correlations among the configurations. Hence, it is both important and challenging to find the best system configuration at runtime.

IV. System Model and Problem Statement

Network and edge service. Our system operates in time slots, each with fixed duration Δ secs. A set $\mathcal{N}$ of N mobile devices are connected to a MEC server that runs a video analytics service, e.g. object recognition, as in Fig. 4. The devices extract images from the captured video stream, where properties like the number and type of objects in each image vary over time. We denote those properties with $\{o_{nt}\}$ which follows an unknown random process $\{o_{nt}\}_{t=1}^{\infty}$. Each user applies an encoding rate to the captured images selected from finite set $\mathcal{X}$ and transmits them to the server for processing. The average signal strength (SNR) of device n during slot t is denoted by h_{nt}, which is calculated by the AP and is given by a random process $\{h_{nt}\}_{t=1}^{\infty}$. Upon reception of an image, the server decodes and downsamples it to fit the input size of the NN that is loaded on its GPU. The possible NN size values are selected from a finite set $\mathcal{Y}$. Note that while each device can apply their own encoding rate, the NN input size is common for all devices as they share the server's GPU[5].

Decision variables. The encoding rate of each device n during time slot t is a system decision variable denoted by $x_{nt} \in \mathcal{X}$. The selection of x_{nt} will determine the resulting image size $s(x_{nt})$, which in turn will affect the transmission time to the server. We denote the fixed bandwidth of the wireless channel by W, and as a result, the Wi-Fi transmission delay of user n during t is:

$$\tau_{nt}(x_{nt}) = \frac{s(x_{nt}) + L}{W \log(1 + h_{nt})}, \tag{1}$$

where L is the TCP/UDP stack overhead added to the images.

To enable multi-user connectivity to the server and coordinate transmissions and GPU computations, we introduce the time allocation variable $w_{nt} \in \mathcal{W} \triangleq [0, \Delta]$ as a configurable parameter. By limiting the fraction of time w_{nt} that is allocated to each device n for continuous object recognition, we can guarantee that all devices have the opportunity to send a number of images during t. We compact all variables in $z_t = (x_{1t}, \ldots, x_{Nt}, y_t, w_{1t}, \ldots, w_{Nt}) \in \mathcal{Z} \triangleq \mathcal{X}^N \times \mathcal{Y} \times \mathcal{W}^N$. Note that our system is orthogonal to, and operates at a higher time scale than other underlying wireless mechanisms, e.g., contention control, which run in the scale of milliseconds.

[5]Our experiments showed that changing the NN input size for each user induces delay that impacts performance. In Sec. VI we extend our model to allow different NN size per user whenever many GPUs are available.

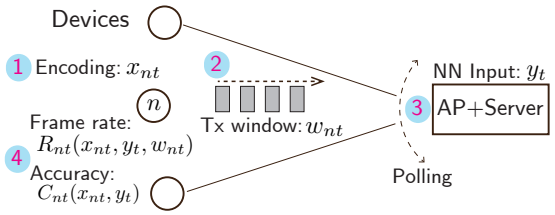


Fig. 4: Each user n applies encoding x_{nt} and sends images for w_{nt} secs to server which has NN size y_t. The user enjoys frame rate R_{nt} and accuracy C_{nt}. The system is configured every Δ secs.

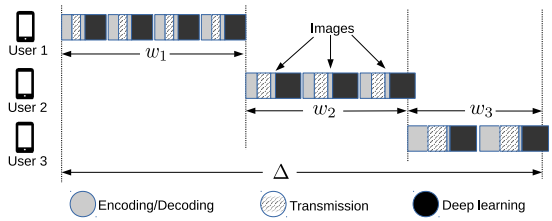


Fig. 5: Task scheduling example for 3 users.

Performance metrics. We define the function $C_n(z_t) : \mathcal{Z} \to \mathbb{R}_+$ to be the expectation of the CC experienced by user n when configuration z_t is applied to the system. In practice however, we can only observe the noisy instantaneous CC achieved during each slot t, that follows a distribution like in Fig 2c. This noise is caused due to the varying content of the images expressed by o_{nt} that makes some objects easier/harder to classify than others. We denote the instantaneous CC as $C_{nt}(z_t; o_{nt}) : \mathcal{Z} \to \mathbb{R}_+,\ n \in \mathcal{N}$, and can write it:

$$C_{nt}(z_t; o_{nt}) = C_n(z_t) + \epsilon_1(o_{nt}), \ \text{ with } \ \epsilon_1 \sim \mathcal{N}(0, \sigma_1^2). \tag{2}$$

In a single user scenario, the frame rate is fully determined by the end-to-end latency of the system. With multiple users however, service is interrupted (see Fig. 5 for a 3-user scheduling example). The frame rate we refer to from now on, is the number of images processed for each user during a slot of length Δ, e.g. with respect to Fig. 5 we have 4 frames per slot for user 1, 3 frames for user 2 and 2 for user 3. Similar to CC, we define the average frame rate of device n by $R_n(z_t) : \mathcal{Z} \to \mathbb{R}_+$, that depends on all variables. Function $R_n(z_t)$ is also an average value that can vary over time due to varying channel conditions h_{nt} of each user, as shown in Fig 2d. We denote the noisy frame rate observed during slot t by $R_{nt}(z_t; h_{nt}) : \mathcal{Z} \to \mathbb{R}_+$ and can write it:

$$R_{nt}(z_t; h_{nt}) = R_n(z_t) + \epsilon_2(h_{nt}), \ \text{ with } \ \epsilon_2 \sim \mathcal{N}(0, \sigma_2^2). \tag{3}$$

Finally, each device n sets a minimum frame rate threshold λ_n based on their preferences or requirements. We consider the general case where these can differ across users.

User scheduling. If we allow the users to concurrently send sequences of images, we face the problem of interference and queuing at the server, since only one image can be processed at each time. In detail, if we consider a shared medium (previous versions of WiFi) we want to avoid the users to collide in their transmissions, i.e., avoid the *contention phase*, which will delay the service

pipeline (encoding-transmission-decoding-processing) shown in Fig. 5. Second, if we consider the latest WiFi standard (802.11ax), which is based on OFDMA, several users can transmit using different subbands without colliding. In that case our aim is to prevent the server queue to grow infinitely. Such problems can deteriorate the analytics performance since the end-to-end latency of a single image can increase, and thus the information overlaid on the user's screen can be *substantially outdated*. To avoid that, we let the server apply a polling scheme, so that each user $n \in \mathcal{N}$ executes the entire processing pipeline (in both the mobile, the network and server) without interruptions, for the duration of its allocated time w_{nt}, using the selected configuration x_{nt}, y_t.

Problem formulation. Our aim is to maximize the CC of users while respecting their frame rate requirements. We define the total observed CC as $f_t(z_t) = \sum_{n\in\mathcal{N}} C_{nt}(z_t; o_{nt})$, and the constraints $g_{nt}(z_t) = \lambda_n - R_{nt}(z_t; h_{nt}),\ n \in \mathcal{N}$. Ideally, we would like to find the optimal solution $z^* = (x_1^*, \ldots, x_N^*, y^*, w_1^*, \ldots, w_N^*)$ to the following problem:

$$\begin{aligned} \mathbb{P}: \underset{z\in\mathcal{Z}}{\text{maximize}} \quad & \mathbb{E}\left\{f_t(z)\right\} && \text{(4a)} \\ \text{subject to:} \quad & \mathbb{E}\left\{g_{nt}(z)\right\} \leq 0,\ n \in \mathcal{N} && \text{(4b)} \\ & \sum_{n\in\mathcal{N}} w_n \leq \Delta && \text{(4c)} \end{aligned}$$

Observe that applying the expectations in (4a)-(4b), by using (2)-(3), yields the unknown average functions, i.e.,

$$\mathbb{E}\left\{f_t(z)\right\} = \sum_{n\in\mathcal{N}} C_n(z), \quad \mathbb{E}\left\{g_{nt}(z)\right\} = \lambda_n - R_n(z).$$

Also, constraints (4b), (4c) ensure that the frame rate thresholds are respected, and that the time allocation is valid.

Clearly, $\mathbb{P}$ cannot be solved directly since we do not have access to functions $C_n(\cdot)$ and $R_n(\cdot)$. Therefore, we follow an *online learning approach* where we select configurations z_t at the beginning of each slot t and calculate the perturbed outputs $f_t(z_t), g_{nt}(z_t)$ using the noisy measurements $C_{nt}(z_t; o_{nt})$, $R_{nt}(z_t; h_{nt})$. Our goal then is to find a sequence of configurations $\{z_t\}_t$ that will drive the average performance close to the benchmark $\mathbb{E}\left\{f_t(z^*)\right\}$, while satisfying (probabilistically) the constraints $g_{nt}(z_t), \forall n$ and (4c) at each slot. Formally, we define the *pseudo-regret*:

$$Reg_T = \sum_{t=1}^{T} \mathbb{E}\left\{f_t(z^*)\right\} - \sum_{t=1}^{T} \mathbb{E}\left\{f_t(z_t)\right\}, \tag{5}$$

and ask that sequence $\{z_t\}_t$ achieves sublinear average regret, $\lim_{T\to\infty} Reg_T/T = 0$. This will ensure that our policy learns to perform as well as the hypothetical benchmark z^* which can only be designed in hindsight, i.e., with complete knowledge of the platform functions and data.

V. Gaussian Processes and Problem Solution

A. MAB formulation through GP modeling

Due to the online nature of our problem, we address it following a Multi-armed Bandit (MAB) approach, by which we sequentially select different *arms* to tackle the exploration-exploitation dilemma. However, most of classic MAB algorithms such as UCB [42] and Thompson Sampling [43], do not consider that nearby arms can be correlated, i.e., they yield similar performance; or assume these correlations to be known in advance, or to have a specific (e.g., linear) structure [44], [45]. Nevertheless, as the experiments in Sec. III showed, the system configurations exhibit unknown, varying and even non-monotonic performance correlations.

In fact, these correlations could be fully characterized by the objective and constraint functions in $\mathbb{P}$, had they been known. To rectify this, we can use Gaussian Processes which is a model-free (or, assumption-free) approach requiring only a certain level of function smoothness [16], something we already validated with our measurements. A kernel function $\rho(z, z')$ is used to express the correlation between the objective/constraint function value of any pair of configurations (z, z') and enables predictions about the function evaluation at any vector $z \in \mathcal{Z}$.

Following this approach, the seminal GP-UCB algorithm [46] was applied to *unconstrained problems* where the objective function is iteratively approximated using noisy observations, much like in our setup with the difference of constraints. The benefit of this approach is that it estimates the mean value of $f_t(z)$ for any z by only using the rewards observed up to t, including configurations that have not been applied in the past. In specific, if $\mathcal{A}_t = \{z_1, \ldots, z_t\},\ \mathcal{F}_t = \{f_1(z_1), \ldots, f_t(z_t)\}$ are the applied configurations and respective rewards up to slot t, the mean value and variance of $f_t(z)$ for any configuration (or, action) z are given by:

$$\mu_{f,t}(z) = \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_1^2 \boldsymbol{I})^{-1} \mathcal{F}_t, \tag{6}$$

$$k_{f,t}(z, z') = \rho(z, z') - \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_1^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(z'), \tag{7}$$

where $\boldsymbol{k}_t(z) = (\rho(z_1, z), \ldots, \rho(z_t, z))^\top$, $\boldsymbol{K}_t = (\rho(z_t, z_{t'}))$ is the positive definite kernel matrix, and $\boldsymbol{I}$ the identity matrix. GP-UCB selects the next action based on a weighted acquisition rule:

$$z_{t+1} = \arg\max_{z\in\mathcal{Z}} \mu_{f,t}(z) + \beta_t \sqrt{k_{f,t}(z, z)},$$

where β_t is a problem-related parameter, and it provably achieves sublinear expected (or, pseudo) regret [46].

B. Constrained GP-based MAB optimization

In order to find configurations that satisfy the frame rate constraints, *and* do so without violating these thresholds, we need a twofold extension of GP-UCB. There are only few works that proposed similar ideas for *safe* GP-UCB algorithms, e.g., [47]–[49]. Following a similar approach, we design a learning algorithm with 2 stages: the *expansion stage* (for T_0 slots) and the *optimization stage* (for $T - T_0$ slots). In the former, given an initial safe set of configurations S_0, i.e., actions guaranteed to satisfy the thresholds, we successively create enlarged safe sets S_t by adding configurations that conservatively (by means of upper bounds) also respect the constraints. After we reach a satisfactory approximation of the maximum achievable safe set, we commence the

Algorithm 1 Automatic configuration of video analytics

Initialize: $S_0 \subset \mathcal{Z}, z_1 \in S_0, \mathcal{A}_0, \mathcal{F}_0, \mathcal{G}_{n0} = \emptyset, \rho(z,z'), M_n, \lambda_n > 0$ and β_t
for $t = 1, \ldots, T$ **do**
  Process images and obtain: $f_t(z_t), g_{nt}(z_t),\ n \in \mathcal{N}$
  $\mathcal{A}_t \leftarrow \mathcal{A}_{t-1} \cup \{z_t\}$
  $\mathcal{F}_t \leftarrow \mathcal{F}_{t-1} \cup \{f_t(z_t)\}$
  $\mathcal{G}_{nt} \leftarrow \mathcal{G}_{nt-1} \cup \{g_{nt}(z_t)\},\ n \in \mathcal{N}$
  Update posteriors of $z \in S_t$ using (6)-(9)
  if $t \leq T_0$ **then**
    $S_t \leftarrow \cap_n \cup_{z \in S_{t-1}} \{z' \in \mathcal{Z} | u_t^n(z) + M_n \|z - z'\|_2 \leq 0\}$
    $G_t \leftarrow \{z \in S_t \mid e_t(z) > 0\}$
    if $\max_{z \in G_t} \left(u_t^n(z) - l_t^n(z)\right) < \zeta, \forall n \in \mathcal{N}$ **then**
      $z_{t+1} \leftarrow \arg\max_{z \in S_t} u_t^f(z)$
    else
      $z_{t+1} \leftarrow \arg\max_{z \in G_t} \left(u_t^n(z) - l_t^n(z)\right), n \in \mathcal{N}$
    end if
  else
    $S_t \leftarrow S_{t-1}$
    $z_{t+1} \leftarrow \arg\max_{z \in S_t} u_t^f(z)$
  end if
end for

optimization stage where we apply the upper confidence bound (UCB) rule on that set, much like in GP-UCB [46].

In detail, we use GPs to model both the constraints, as we do for the objective, and evaluate their posteriors using the past observations $\mathcal{G}_{nt} = \{g_{n\tau}(z_\tau)\}_{\tau=1}^t$ as:

$$\mu_{n,t}(z) = \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_2^2 \boldsymbol{I})^{-1} \mathcal{G}_{nt}, \tag{8}$$

$$k_{n,t}(z,z') = \rho(z,z') - \boldsymbol{k}_t(z)^\top (\boldsymbol{K}_t + \sigma_2^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(z'). \tag{9}$$

We also use the upper and lower confidence bounds (UCBs, LCBs) for the constraint and objective functions:

$$u_t^i(z) = \mu_{i,t}(z) + \beta_t \sqrt{k_{i,t}(z,z)},\ i = f, 1, \ldots, N \tag{10}$$

$$l_t^i(z) = \mu_{i,t}(z) - \beta_t \sqrt{k_{i,t}(z,z)},\ i = f, 1, \ldots, N \tag{11}$$

where β_t is an increasing with t scalar (discussed below).

Regarding the safe set expansion stage, if we knew the constraints it could be achieved by performing the operation:

$$V_\zeta(S_t) = S_t \cup \bigcap_{n \in \mathcal{N}} \Big\{ z \in \mathcal{Z} \ \big| \exists\ z' \in S_t : g_n(z') + \zeta + M_n \|z - z'\|_2 \leq 0 \Big\},\ t = 0, 1, 2 \ldots$$

where M_n is the Lipschitz constant of g_n, and ζ a tunable tolerance parameter. Essentially, we would expand S_t by including points z that are close enough to previous safe points z' such that they also satisfy the constraints. We denote with $S_{max}^\zeta \triangleq \lim_{t \to \infty} V_\zeta(S_t)$ the *maximum reachable* set through this operation, and S_{max} the *maximum possible* safe set that we obtain for $\zeta = 0$. Yet, since we do not know the constraints we follow a different approach.

Namely, we use instead the UCBs and the expansion rule:

$$S_t = \bigcap_{n \in \mathcal{N}} \bigcup_{z \in S_{t-1}} \Big\{ z' \in \mathcal{Z} \mid u_t^n(z) + M_n \|z - z'\|_2 \leq 0 \Big\} \tag{12}$$

and employ the updated safe set S_t to create a second set $G_t \subseteq S_t$ that contains configurations which not only are safe but can lead to further expansion. For that, we define:

$$e_t(z) = \Big| \bigcap_{n \in \mathcal{N}} \big\{ z' \in \mathcal{Z} \backslash S_t \ \big|\ l_t^n(z) + M_n \|z - z'\|_2 \leq 0 \big\} \Big|, \tag{13}$$

and then build $G_t = \{z \in S_t \mid e_t(z) > 0\}$. Finally, if the configurations in G_t are still uncertain enough in terms of their possible values, i.e. $\max_{z \in G_t} (u_t^n(z) - l_t^n(z)) \geq \zeta, \forall n$, we select the most uncertain $z_{t+1} = \arg\max_{z \in G_t} (u_t^n(z) - l_t^n(z))$. Otherwise, we select the configuration with the highest UCB, i.e., $z_{t+1} = \arg\max_{z \in S_t} u_t^f(z)$. In that case, we have found a good approximation for the safe set, i.e., close enough to the maximum reachable set S_{max}^ζ, and can continue in the optimization stage. All steps are shown in Algorithm 1.

C. Theoretical results

The effectiveness of Algorithm 1 relies on the accurate estimation of sets S_t and G_t. Specifically, we want to conservatively expand the safe set in order to guarantee the feasibility of its configurations. On the other hand, if the expansion is too conservative, we will need many iterations to reach the set of all safe configurations S_{max}^ζ. This trade off is controlled by parameter β_t which is chosen as [48]:

$$\beta_t = B + \sigma_1 \sqrt{2(1 + \gamma_{t-1} + \log(1/\delta))}, \tag{14}$$

In the above, B is an upper bound on the Reproductive Kernel Hilbert Space (RKHS) norm of f and g_n, while δ is the allowed *constraint violation probability*. Parameter γ_t is the maximum mutual information gain that can be obtained about the prior of f, after t samples have been observed [46]:

$$\gamma_t = \max_{A \subset Z, |A| = t} = \frac{1}{2} \log |\boldsymbol{I} + \sigma_1^{-2} \boldsymbol{K}_A|,$$

where $\boldsymbol{K}_A = [\rho(z,z')],\ z, z' \in A$ is the covariance matrix of the samples collected after t slots. Evidently, γ_t is very difficult to obtain in practice, but a conservative bound is given in [47] for the case of finite $\mathcal{Z}$ as

$$\gamma_t \leq |\mathcal{Z}| \log \left(1 + \sigma_1^{-2} t |\mathcal{Z}| \max_{z \in \mathcal{Z}} k_{f,t}(z,z)\right).$$

We employ the Matern kernel function with parameter $\nu = 3/2$, which implies that our functions are at least once differentiable [16]. The kernel is given by:

$$\rho(z,z') = \Big(1 + \frac{\sqrt{3}}{l} \|z - z'\|_2\Big) \Big(\exp(-\frac{\sqrt{3}}{l} \|z - z'\|_2\Big),$$

where l is a length scale parameter.

Next, we formally present the convergence properties of the safe set (expansion stage) and the average observed reward (optimization stage), to S_{max}^ζ and $\mathbb{E}\{f_t(z^*)\}$, respectively. For the former, what we need to do is find the minimum T_0 in the problem's time horizon T that guarantees this convergence. This is described as follows:

Lemma 1. *Given an initial safe set $S_0 \neq \emptyset$ such that $g_n(z) \leq 0,\ \forall z \in S_0, n \in \mathcal{N}$, fix any $\zeta > 0$ and $\delta \in (0,1)$, choose β_t as in* (14)*, and $\gamma_t = |\mathcal{Z}| \log(|\mathcal{Z}| t)$. The safe set expansion*

stage of Algorithm 1 guarantees with probability $1-\delta$ *that only safe actions are included to the safe set at any time. Moreover, the expanded set* S_t *will reach the maximum safe set* S^{ζ}_{max} *if we select* T_0 *to be the smallest integer for which:*

$$\frac{T_0}{\beta^2_{T_0}|\mathcal{Z}|\log(|\mathcal{Z}|T_0)} \geq \frac{8(|S_{max}|+1)}{\zeta^2\log(1+\sigma_1^2)}.$$

Proof. The proof is based on Theorem 1 in [48] where we apply the bound on the information gain γ_t. This is possible since in our setup the action set $\mathcal{Z}$ is always finite. ■

The next theorem characterizes the algorithm's convergence, and how its regret depends on the system parameters.

Theorem 1. *Given an initial safe set* $S_0 \neq \emptyset$ *such that* $g_n(z) \leq 0 \ \forall z \in S_0$, $n \in \mathcal{N}$, *fix* $\delta \in (0,1)$, *and choose* β_t *as in* (14). *Algorithm 1 yields sublinear regret of* $\mathcal{O}(\sqrt{T}|\mathcal{Z}|\log(|\mathcal{Z}|T))$ *with probability* $1-\delta$. *In specific:*

$$Reg_T \leq 4B\sqrt{(T+2)\gamma_T} + \gamma_T\sqrt{(T+2)(\alpha/\gamma_t+1)},$$

where $\alpha = 1+\log(1/\delta)$, *and* $\gamma_T = |\mathcal{Z}|\log(|\mathcal{Z}|T)$.

Proof. By the definition of regret we have

$$Reg_T = \sum_{t=1}^{T}\mathbb{E}\{f_t(z^*)\} - \mathbb{E}\{f_t(z_t)\} \leq \sum_{t=1}^{T}\mu_{f,t}(z_t) + \beta_t\sqrt{k_{f,t}(z_t,z_t)} - \mathbb{E}\{f_t(z_t)\} \leq 2\beta_T\sum_{t=1}^{T}\sqrt{k_{f,t}(z_t,z_t)}$$

where we used the upper and lower bounds (10),(11) and the fact that β_t is an increasing parameter. From Lemma 4 in [50] we have that $\sum_{t=1}^{T}\sqrt{k_{f,t}(z_t,z_t)} \leq \sqrt{4(T+2)\gamma_T}$ hence we obtain

$$\begin{aligned} Reg_T &\leq 2\beta_T\sqrt{4(T+2)\gamma_T} \\ &\leq 4\big(B+\sqrt{2}\sigma_1\sqrt{\alpha+\gamma_T}\big)\sqrt{(T+2)\gamma_T} \\ &= 4B\sqrt{(T+2)\gamma_T} + \gamma_T\sqrt{(T+2)(\alpha/\gamma_t+1)}. \end{aligned}$$

Observe that the largest (second) term of the bound yields a regret growth of $\mathcal{O}(\sqrt{T}\gamma_T)$ and by the selection of γ_T we have $\mathcal{O}(\sqrt{T}|\mathcal{Z}|\log(|\mathcal{Z}|T))$. ■

Discussion. The above result shows that the cumulative regret does not grow indefinitely and the algorithm selects configurations towards increasing the obtained rewards. The performance of the algorithm depends on parameters such as ζ which allow us to set the optimization accuracy – increasing it reduces the expansion time T_0 but shrinks the range of considered configurations (by the algorithm and the benchmark); while reducing parameter δ improves the violation and regret bound probabilities but deteriorates the regret bound. Finally, note that all bounds are probabilistic, hence the term *pseudo*-regret.

VI. Extensions & Practical Considerations

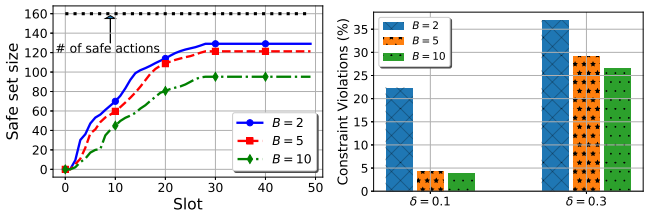
Next, we present important extensions of our system model, and also describe implementation issues that allow the practical deployment of Algorithm 1.

Transmission control and sequencing. Besides scheduling the users, in many scenarios it is crucial to guarantee a low maximum delay between consecutive schedulings of each user. In Fig. 5 for example, this maximum delay is equal to $w_2+w_3 = \Delta - w_1$ for user 1. A way to reduce this delay is to divide the slot into smaller subintervals, e.g. k sub-slots of duration Δ/k, which will effectively reduce the inter-arrival delay by a factor of k. Alternatively, one might resort to interleaving transmissions for users with high performance requirements, and break the scheduling pattern. Moreover, note that this framework operates at a higher time scale than typical wireless mechanisms, e.g. power allocation, which run in a much smaller time scale. These decisions are orthogonal to the video pipeline configuration, and are essentially latent factors, the effect of which is incorporated through our Bayesian updates.

Additional configurations. In our prototype we experimented with the NN size, encoding rate and airtime. Nevertheless, other video processing pipelines involve parameters such as the frame resolution [10], [14], or NN model and number of NN layers [13], [30], [35]. These parameters eventually trade off frame rate for CC, just like the encoding rate and NN input size in our application, and our framework can be readily extended to account for these options. For example, consider the case where we can select the users' frame resolution p_{nt} from a finite set $\mathcal{P}$, on top of the encoding rate. The impact on image size $s(x_{nt},p_{nt})$ and transmission delay $\tau_{nt}(x_{nt},p_{nt})$ would be 2-dimensional and the configuration vector would be $z_t = (x_{1:Nt}, y_t, w_{1:Nt}, p_{1:Nt})$, where we use shorthand notations $\alpha_{1:Nt}$ for vectors $(\alpha_{1t},\ldots,\alpha_{Nt})$. Similarly, if we can select among $\mathcal{L}$ NN models that differ on, e.g., their training data, this vector becomes $z_t = (x_{1:Nt}, y_t, w_{1:Nt}, l_t), l_t \in \mathcal{L}$. Such extensions increase the configuration space and this can impact the convergence speed, which nevertheless is guaranteed.

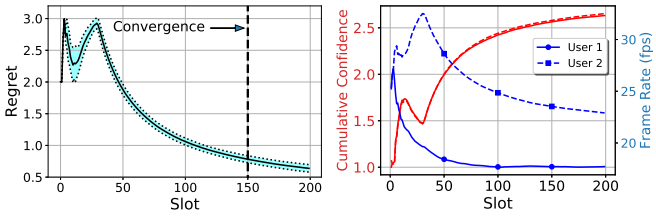
Controlling computing resources. On the other hand, some systems offer access to allocating their computing resources or have multiple GPUs. Hence, we would be able to allocate a GPU, and as a result a distinct NN input size y_{nt} for each user n. The cost would be again an increased action space, namely $z_t = (x_{1:Nt}, y_{1:Nt}, w_{1:Nt})$. Furthermore we can introduce assignment variables to allocate multiple users to multiple GPUs and/or Access Points (AP). In specific, consider that the server has K available GPUs and the users can connect to it through J APs, resulting in a joint GPU/AP assignment decision vector, i.e. $z_t = (x_{1:Nt}, y_{1:Kt}, w_{1:Nt}, v_{1:Nt})$, where v_{nt} is the association decision for user n, i.e. a tuple (j,k) that denotes n is served by AP j and GPU k. This way we can support high frame rates for the users since the resource availability scales. We evaluate these scenarios in Sec. VII.

Implementation issues. In many settings some of the algorithm's parameters might be unknown. For instance, an



(a) (b)

Fig. 6: (a) Safe set expansion during the first stage of Algorithm 1. (b) Constraint violation percentage for different values of B, δ.



(a) (b)

Fig. 7: (a) The average regret of Algorithm 1. (b) Cumulative Confidence and frame rate of 2 users with $\lambda_1 = 10$, $\lambda_2 = 20$.

upper bound for the norms of f, g_n is difficult to compute with no/incomplete data. The same is true regarding the Lipschitz constants M_n. In practice, we can compute the former during a small initialization period, or rely on historic data. For the latter we can use a modified rule for the expansion stage [48], where we replace (12), (13) with:

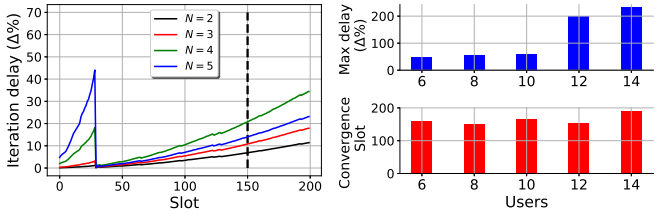
$$S_t = \bigcap_{n\in\mathcal{N}} \{z \in \mathcal{Z} \mid u_t^n(z) \leq 0\}$$
$$e_t(z) = \Big| \bigcap_{n\in\mathcal{N}} \{z' \in \mathcal{Z}\backslash S_t \mid l_t^n(z) \leq 0\} \Big|,$$

where we simply use the upper/lower confidence bounds. The drawback is that we need to calculate the posteriors for all $z \in \mathcal{Z}$, not just the ones already in S_t.

Another important aspect is that the execution duration (in slots) of both the expansion and optimization stages cannot be set a-priori; hence, a stopping criterion should be employed in practice. The expansion stage can be terminated if the safe set does not increase for, e.g., 10 consecutive slots with a hard cap of 30 slots, as discussed also in [48]. The optimization stage can be terminated if we do not observe a further increase in the reward for a fixed number of, e.g., 10 slots. Clearly, these rules depend also on how fast the server can actually execute the algorithm iterations, where each one needs to be completed within Δ seconds, i.e., before applying the next configuration. The complexity of Algorithm 1 is dominated by the inversion operation of the GPs, which is in the order of $O(N^3)$ for N data points [16]. Hence, as time evolves these computations become more cumbersome. Our experiments show that a typical server can execute them well-before the Δ secs window expires (see Sec. VII), while this delay can be tuned with the above termination rules.

VII. Performance Evaluation

We consider the sets $\mathcal{X} = \{25, 50, 75, 100\}$, and $\mathcal{Y} = \{128, 192, \ldots, 576\}$. We quantize the time allocation decisions w_{nt} so that our configuration space $\mathcal{Z}$ is finite. In specific, we define $\mathcal{W} = \{.1, .2, .3, .4, .5, .6, .7, .8, .9\}$ and $\Delta = 5$ sec, so that $w_{nt} = 0.5$ means that the time allocated to device n during t is 2.5 sec. For the construction of the initial safe set S_0, we only use configurations that include the lowest NN size and encoding rate, i.e. 128 and 25% respectively, since if the problem is feasible, these parameters will definitely satisfy the constraints.



(a) (b)

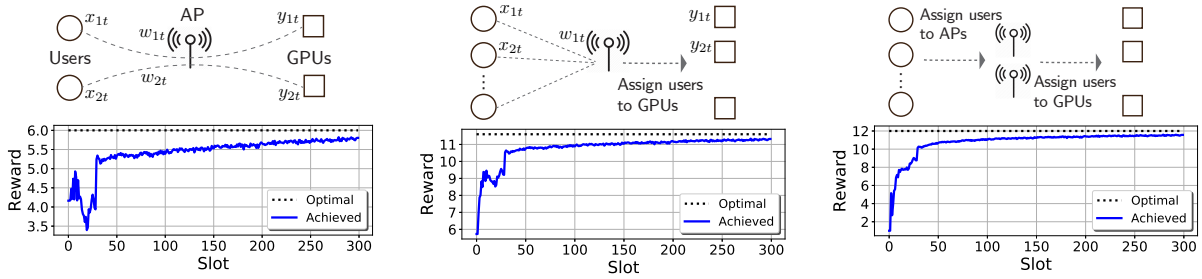
Fig. 8: (a) Average iteration delay of Alg. 1. (b) Maximum iteration delay and convergence time in slots.

We combined the above measurements with the model of Sec. IV to evaluate Algorithm 1 in finding the optimal configuration of a multi-user system with diverse frame rate constraints. The channel bandwidth is $W = 40$ MHz and each user's mean SNR is selected from a uniform distribution in $[10, 35]$ dB. This mean is then used to sample the SNR h_{nt} at each slot from a Gaussian distribution. All our results below are averages from 100 evaluations.

Parameter Analysis. We first study the impact of parameter β_t. The value for B impacts the safe set expansion stage since it controls how conservative or slack we are in adding configurations to the safe set. In addition, parameter δ determines the constraint violation probability which is related to the correctness of S_t and how likely it is for relatively unsafe actions to be selected. Fig. 6a depicts the size evolution of the safe set over time versus B. We calculated (offline) that the number of configurations that satisfy $g_n(z) \leq 0, \ \forall n$ is 160. We observe that as we increase B the algorithm becomes more conservative in expanding the safe set. In specific, we have that $|S_t|$ for $B = 2$ is 80.6% of the actual safe set, while for $B = 10$ it is only 59.4%, meaning that many high reward actions will not be considered in the optimization stage.

Next, we evaluate the impact of B and δ on the constraints violation probability. Fig. 6b displays the constraint violation probability over 200-slot simulations. We consider probabilities 0.1 and 0.3 for δ, and $B \in \{2, 5, 10\}$, as before. Notice that for $B = 2$ the violation probability increases beyond 10% and 30% respectively, indicating that the selection of B is too low to satisfy the desired probability. In the following we select $\delta = 0.1$ and $B = 5$.

Results. We evaluate the performance of Algorithm 1 using the average regret $\bar{r}_t$, for a 2-user system where



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 9: Reward of (a): Many GPUs and preassigned users; (b): User-to-GPU assignment; (c): User-to-AP-to-GPU assignment.

$\lambda_1 = 10, \lambda_2 = 20$ fps. In Fig. 7a we depict the average regret, where the light blue shaded area indicates the 1-std area over 100 evaluations. The figure highlights that (after the safe set expansion stage) the algorithm makes high reward actions, resulting in a continuous decrease of $\bar{r}_t$ that is asymptotically 0. We impose the stopping criterion discussed in Sec. VI and observe that convergence roughly occurs after 150 slots.

In order to evaluate the performance of each user, we plot their achieved average CC and frame rate over time in Fig. 7b. The figure shows $1/t \sum_{k=1}^{t} C_{nk}(z_k; o_{nk})$ and $1/t \sum_{k=1}^{t} R_{nk}(z_k; h_{nk}),\ \forall t,\ n = 1, 2$, respectively in each of the y-axes. Observe that during the expansion stage, i.e. $t \leq 30$, we have a rather random performance since the goal there is only to locate safe actions. For $t > 30$ however, the algorithm takes improved actions for both users, resulting in an increasing CC. These actions are at the edge of the safe set and hence they are "riskier" resulting in a controlled drop of the average frame rate, which is always above each user's threshold λ_n. Additionally, the achieved CC is almost identical for both users, which indicates that the differentiation in time allocation rather than encoding rate is what differentiates the users' frame rates, since the former does not affect the CC.

Next, we evaluate the algorithm's scalability by measuring its average iteration delay, and in particular, the time required to execute steps 7-19 in our server (see Sec. III for server specs). Fig. 8a depicts this delay as a fraction of slot duration Δ for different number of users N. For the first 30 slots (expansion stage) we clearly see the delay increasing both with the slot t and users N. The former is because the updates (6)-(9) increase in complexity with the samples (at cubic rate), since they involve matrix inversions of size t. The latter is due to the fact that with more users, there are many more candidate configurations for the safe set expansion. After the first stage, we observe a drop of the delay since *(i)* the posteriors of the constraint functions no longer require updates, and *(ii)* the safe set has been fixed and $u_t^f(z)$ is only evaluated for $z \in S_t$. The iteration delay starts increasing again with t for the same reason as before, but is kept low until the algorithm converges to an acceptable solution. Interestingly, the delay for $N = 4$ is bigger than with $N = 5$, which is due to the smaller $|S_t|$ we get for the latter case. Namely, the more users we have the harder it gets to satisfy their frame rate constraints, which in turn might shrink the safe set and expedite the algorithm.

We consider more users in Fig. 8b where we set a low frame rate requirement $\lambda_n = 2,\ \forall n \in \mathcal{N}$, so that problem $\mathbb{P}$ is feasible. In the top graph we show the maximum value of the iteration delay within a 200 slots evaluation. Notice that for $N \geq 12$ the delay gets much bigger than the slot duration, which suggests that we have to either increase Δ and admit longer convergence, or reduce the expansion stage duration. The lower graph in Fig. 8b shows the slot in which (on average) the stopping criterion discussed in Sec. VI occurs for different values of N. We observe that the differences are insignificant and that we can always stop the algorithm in fewer than 200 slots.

Finally, we evaluate our framework for the settings where *(i)* multiple GPUs ($K = 2$) are available to the server, and a NN size configuration y_{nt} is selected for each user n (Scenario 1); *(ii)* the number of users N is higher than the number of GPUs K (Scenario 2); and *(iii)* the users can be served by a number of J APs. In detail we set $N = 4, K = 2, J = 1$ for Scenario 2, and $N = 4, K = 2, J = 2$ for Scenario 3. The achieved and optimal reward of Algorithm 1 for Scenarios 1-3 is displayed in Fig. 9a-9c, along with a small diagram depicting each setting. Remember that the achieved reward is simply the added observed CC for all users in each slot. We can see that in all scenarios the performance of the system keeps increasing and converging towards the optimal one. In specific, the observed performance is within only $6\%, 4\%$ and 5% from the optimal in each Scenario, after 200 slots.

VIII. Conclusions

Using an exemplar prototype, we demonstrated that MEC-assisted video analytics exhibit volatile and platform/data-dependent performance. Our framework makes no assumptions on the form of objective and constraint functions, and is inspired by ideas in the area of automated machine learning, which we extend here in order to configure the network too. Our solution firstly identifies feasible configurations, and then finds a sequence of actions that converge to the problem's optimal solution. We believe that our work paves the road for building systems that are fully adaptable and also provide performance guarantees.

Acknowledgments

This publication has emanated from research supported by SFI grants 17/CDA/4760, 16/IA/4610 and 17/NSFC/5224.

REFERENCES

[1] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile Augmented Reality Survey: From Where We Are to Where We Go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[2] Microsoft, "Seeing AI," https://www.microsoft.com/en-us/ai/seeing-ai.

[3] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf, "Deepface: Closing the Gap to Human-level Performance in Face Verification," in *Proc. of IEEE CPVR*, 2014.

[4] W. Zhang, B. Han, and P. Hui, "On the Networking Challenges of Mobile Augmented Reality," in *Proc. of ACM SIGCOMM Workshop on VR/AR Network*, 2017.

[5] Y. C. Hu *et al.*, "Mobile Edge Computing - a Key Technology Towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[6] M. Jia, et al., "Cloudlet Load Balancing in Wireless Metropolitan Area Networks," in *Proc. of IEEE INFOCOM*, 2016.

[7] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online Job Dispatching and Scheduling in Edge-Clouds," in *Proc. of IEEE INFOCOM*, 2017.

[8] X. Chen, L. Jiao, W. Li, X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[10] J. Jiang *et al.*, "Chameleon: Scalable Adaptation of Video Analytics," in *Proc. of ACM SIGCOMM*, 2018.

[11] C.-C.Hung,et al., "VideoEdge:Processing Camera Streams Using Hierarchical Clusters," in *Proc. of IEEE/ACM SEC*, 2018.

[12] P. Yang *et al.*, "Edge coordinated query configuration for low-latency and accurate video analytics," *IEEE Trans. on Industrial Informatics*, vol. 16, no. 7, pp. 4855–4864, 2020.

[13] X. Ran *et al.*, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *Proc. of IEEE INFOCOM*, 2018.

[14] H. Zhang *et al.*, "Live Video Analytics at Scale with Approximation and Delay-Tolerance," in *Proc. of USENIX NSDI*, 2017.

[15] A. Galanopoulos, V. Valls, G. Iosifidis, and D. J. Leith, "Measurement-driven Analysis of an Edge-Assisted Object Recognition System," in *Proc. of IEEE ICC*, 2020.

[16] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[17] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. of ICML*, 2010.

[18] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, "Algorithms for Hyper-Parameter Optimization," in *Proc. of NIPS*, 2011.

[19] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," in *Proc. of ACM SIGKDD KDD*, 2013.

[20] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. d. Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[21] A. Galanopoulos, "Edge-Dataset Description." [Online]. Available: https://github.com/apgalano/Edge-Dataset

[22] C. Lo, Y. Su, C. Lee, and S. Chang, "A Dynamic Deep Neural Network Design for Efficient Workload Allocation in Edge Computing," in *Proc. of IEEE ICCD*, 2017.

[23] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices," in *Proc. of IEEE ICDCS*, 2017.

[24] W. Zhang, B. Han, and P. Hui, "Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking," in *Proc. of ACM Conference on Multimedia*, 2018.

[25] T. Y.-H. Chen *et al.*, "Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices," in *Proc. of ACM SenSys*, 2015.

[26] P. Jain, J. Manweiler, and R. Roy Choudhury, "OverLay: Practical Mobile Augmented Reality," in *Proc. of ACM MobiSys*, 2015.

[27] L. Liu, H. Li, and M. Gruteser, "Edge Assisted Real-time Object Detection for Mobile Augmented Reality," in *Proc. of ACM MobiCom*, 2019.

[28] H. Li *et al.*, "JALAD: Joint Accuracy- and Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution," in *Proc. of IEEE ICPADS*, 2018.

[29] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "MobiQoR: Pushing the Envelope of Mobile Edge Computing Via Quality-of-Result Optimization," in *Proc. of IEEE ICDCS*, 2017.

[30] Q. Liu, et al., "An Edge Network Orchestrator for Mobile Augmented Reality," in *Proc. of IEEE INFOCOM*, 2018.

[31] T. Tan, and G. Cao, "FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile," in *Proc. of IEEE INFOCOM*, 2020.

[32] S. Yi *et al.*, "LAVEA: Latency-aware Video Analytics on Edge Computing Platform," in *Proc ACM/IEEE SEC*, 2017.

[33] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proc. of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[34] Q. Liu, and T. Han, "DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing," in *Proc. of IEEE ICNP*, 2018.

[35] C. Wang *et al.*, "Joint Configuration Adaptation and Bandwidth Allocation for Edge-based Real-time Video Analytics," in *Proc. of IEEE INFOCOM*, 2020.

[36] J. Meng, et al., "Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach," in *Proc. of IEEE INFOCOM*, 2019.

[37] ——, "Dedas: Online Task Dispatching and Scheduling with Bandwidth Constraint in Edge Computing," in *Proc. of IEEE INFOCOM*, 2019.

[38] T. Lin *et al.*, "Microsoft COCO: Common Objects in Context," *arXiv*, vol. abs/1405.0312, 2014.

[39] M. Feurer, et al., "Efficient and Robust Automated Machine Learning," in *Proc. of NIPS*, 2015.

[40] O. Alipourfard et al., "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics," in *Proc. of USENIX NSDI*, 2017.

[41] Q. Liu, and T. Han, "VirtualEdge: Multi-Domain Resource Orchestration and Virtualization in Cellular Edge Computing," in *Proc. of IEEE ICDCS*, 2019.

[42] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-Time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 23, pp. 235–256, 2002.

[43] H. Junya and T. Akimichi, "Optimality of Thompson Sampling for Gaussian Bandits Depends on Priors," *arXiv preprint arXiv:1311.1894*, 2013.

[44] S. Filippi, O. Cappe, A. Garivier, and C. Szepesvári, "Parametric Bandits: The Generalized Linear Case," in *Proc. of NIPS*, 2010.

[45] Y. Russac, O. Cappé, and A. Garivier, "Algorithms for Non-Stationary Generalized Linear Bandits," *arXiv preprint arXiv:2003.10113*, 2020.

[46] N. Srinivas *et al.*, "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting," *IEEE Trans. on Information Theory*, vol. 58, no. 5, pp. 3250–3265, 2012.

[47] Y. Sui, A. Gotovos, J. W. Burdick, and A. Krause, "Safe Exploration for Optimization with Gaussian Processes," in *Proc. of ICML*, 2015.

[48] Y. Sui, V. Zhuang, J. W. Burdick, and Y. Yue, "Stagewise Safe Bayesian Optimization with Gaussian Processes," in *Proc. of ICML*, 2018.

[49] S. Amani, M. Alizadeh, and C. Thrampoulidis, "Regret Bounds for Safe Gaussian Process Bandit Optimization," *arXiv preprint arXiv:2005.01936*, 2020.

[50] S. R. Chowdhury and A. Gopalan, "On Kernelized Multi-Armed Bandits," in *Proc. of ICML*, 2017.