Bayesian Online Learning for MEC Object Recognition Systems

Apostolos Galanopoulos*, Jose A. Ayala-Romero*, George Iosifidis*, Douglas J. Leith*
* School of Computer Science and Statistics, Trinity College Dublin, Ireland

***Abstract*—Real-time object recognition is becoming an essential part of many emerging services, such as augmented reality, which require accurate inference in a timely fashion with low delay. We consider an edge-assisted object recognition system that can be configured in ways that have diverse impacts on these key performance criteria. Our goal is to design an online algorithm that learns the optimal configuration of the system by observing the outcomes of configurations applied in the past. We leverage the structure of the problem and combine a gaussian process with a multi-armed bandit framework to efficiently solve the problem at hand. Our results indicate that our solution makes better configuration choices compared to other bandit algorithms, resulting in lower regret.**

***Index Terms*—Multi-Armed Bandits, Edge Computing, Real-Time Object Recognition.**

I. Introduction

A. Background and Motivation

Recent advances in wireless networks and machine learning are enabling edge intelligence, whereby powerful servers deployed in close proximity to mobile users run compute-intensive Neural Networks (NN) to perform real-time inference on users' sensed data [1]. In particular, Deep Learning (DL) based object recognition is key to future services such as autonomous driving [2] and augmented reality apps [3].

These systems require low end-to-end latency and accurate inference [3]. Numerous parameters affect these performance metrics, e.g. bandwidth and Multi-Access Edge Computing (MEC) resource availability, but many of these are essentially fixed. Two of the main configurable parameters are the DL model used (in particular the size of the NN) and the quality of transmitted images (which can be adjusted via the image encoding rate) [4], [5], and it is these that we focus on here. It is, however, challenging to find the optimal configuration settings since the performance of each configuration is unknown a-priori, and can change depending on factors like the environment conditions and the hardware specifications of mobile devices, which can vary. Also, there is often no analytical expression for the performance under any configuration, and acquiring one requires gathering prohibitively large amounts of data, compromising the system's real-time performance.

B. Related Work

While numerous studies have investigated optimizing the performance of edge computing systems [3], the majority focus on throughput, delay, and other general criteria. With a few notable exceptions [5]–[7], most of these previous studies also do not consider actual services and do not evaluate operation using experimental measurements. In [5]–[7] smart offloading tactics and object tracking are used to improve performance, and online learning of the optimal system parameter configuration is not considered.

Perhaps the closest work to the present paper is that of [4], [8], [9]. DeepDecision [4] selects the frame rate, video encoding rate and DL model to optimize frame rate and accuracy; VideoStorm [8] schedules GPU resources to maximize an accuracy/latency utility; and Chameleon [9] decides the NN model, frame rate, and image resolution to maximize accuracy (not frame rate or latency). However, these all assume that the accuracy and delay of the available DL models and wireless links are known a-priori. In practice however, quite often we do not have access to complete datasets, and even then, their validity is limited as they presume a stationary environment. In this paper we follow a data-driven approach where we make minimal assumptions about the modeling of optimization task.

Most works applying online learning to MEC systems are focused on making offloading decisions, e.g. [10]. Recently, online learning has been applied for configuring MEC systems for real-time operation, without however considering accuracy [11]. In [12] a Lyapunov framework is used to configure the image resolution and frame rate of video analytics applications in an online fashion. The objective is to optimize the accuracy/energy consumption trade-off, under latency constraints. In [13] video quality and computing resources are selected to maximize the (approximate) accuracy, without however considering the frame rate.

The problem we consider here differs from this previous work since most of them handle the frame rate as a configurable parameter (if at all), rather than an optimization objective. Moreover, we leverage a Multi-Armed Bandit (MAB) framework [14] that exploits the correlation between actions, allowing us to quickly find near-optimal configurations.

C. Methodology and Contributions

We design an online algorithm to learn the system configuration, i.e. the NN size and image encoding rate that optimizes the mobile user performance. Extending our initial measurements in [15], we find that there is a non-trivial trade-off between accuracy and latency (or equivalently the frame rate). Hence, we design a unified performance metric that combines the requirements for accurate DL inference *and* a frame rate that is close to a desired target, without however imposing a strict requirement. We propose a MAB framework to tackle the unavailability of analytical expressions for the system's performance. Moreover, we find that similar actions

perform closely, and leverage this correlation using a model-free, Bayesian framework to efficiently explore the action space and achieve near-optimal performance.

The contributions of our paper are summarized as follows:

- We build an example adaptive object recognition system and use real data to evaluate its performance. Our measurements reveal trade-offs between important optimization criteria (accuracy, latency), as well as correlations between the possible system configurations.
- We formulate a system configuration (NN size and encoding rate) optimization problem, and discuss its properties. This data-driven formulation allows us to develop accurate and practical optimization models.
- We leverage a Gaussian Process (GP) model to capture the performance similarity between configurations. This allows for a practical Bayesian approach in modeling the system performance function, which is initially unknown.
- We further combine GPs with a MAB framework to design a no-regret algorithm to configure the system.
- We evaluate the performance of our solution against typical MAB algorithms and find that it can select optimal configurations up to 32% more often.

The rest of the paper is organized as follows. Sec. II describes the design of our MEC system and the reasons that enable an online solution. In Sec. III we formulate our problem and propose its online solution, while in Sec. IV we present the experimental evaluation of the proposed solution. Finally, Sec. V concludes the paper.

II. System Design and Measurements

A. System Setup and Configurable Parameters

Our system is comprised of a mobile client running an android application for capturing, encoding and transmitting images; and a GPU-enabled server performing real-time object recognition on the received images (see Fig.1). The operation of the system is as follows. The mobile device captures an image and performs JPEG compression at a certain encoding rate q. This rate denotes the ratio of the size of the compressed image over the size of the uncompressed one. The resulting image file is transmitted through WiFi to the server, that performs (i) JPEG decoding, (ii) image rotation (if necessary) and (iii) object recognition using YOLO [16], a state-of-the-art object recognition system. The results of the DL computations are the image coordinates of the bounding boxes of recognized objects, along with their respective labels and recognition confidence. Those, are forwarded back to the mobile and overlayed on the screen.

YOLO takes a $n \times n$ image as input and down-samples it by 32 to produce a $n/32$ grid, which is used to detect objects. The selection of n will impact the accuracy of predictions, as well as the delay of processing an image. Naturally, as n increases, the predictions are more accurate but take longer to produce. Similar behavior is revealed by our measurements for the second configurable parameter of the system, the encoding rate q. The larger it is, the more accurate predictions can be, due to the higher quality of the resulting images.

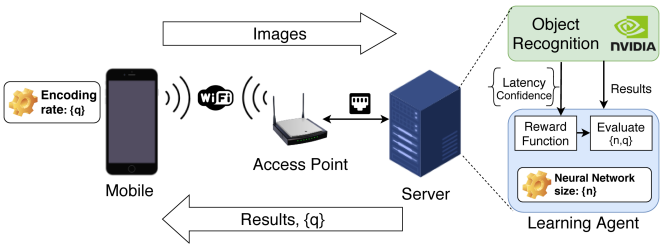


Fig. 1: Architecture of the MEC object recognition system.

However, encoding in high rates takes longer, and so does transmitting a larger file over WiFi.

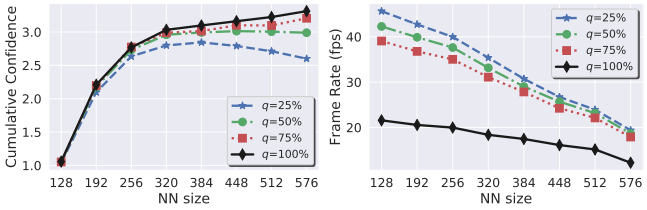
B. Online Learning Approach Motivation

The trade-offs between accurate predictions and frame rate were first explored in our previous work [15]. In that work however, the optimal configuration is computed offline, using a dataset with 5000 cross-validation images. The acquisition of such datasets can be costly in terms of time and, in addition, the gathered data depend on the specific edge application setup (i.e., edge server, GPU model, network delay, etc.). Moreover, since the system is optimized offline, we cannot take into account changes in the congestion of the network or the available bandwidth, that ultimately affect the delay.

For these reasons, we propose an *Online Learning* framework that dynamically selects system configurations and measures their performance. Our algorithm learns from experience aiming at maximizing the accuracy and frame rate (measured by the edge service without using models). Therefore, our solution works in a *plug-and-play* manner without requiring any parametric model or offline gathered data [17].

In this context, our framework has to face the exploration versus exploitation dilemma, i.e., to find a balance between exploring new configurations to find profitable ones (but taking the risk of selecting low-performance ones) or exploiting the configurations with the best empirical performance so far. This dilemma is captured in the MAB problem for which many algorithms have been proposed in the literature [14].

One of the basic assumptions of the classic MAB problem formulation is that the rewards associated with each configuration are independent and identically distributed (i.i.d.). However, this assumption does not hold in our setting. Specifically, we observe based on our experiments, that the closer two encoding rate values are, the closer the size of their compressed images are, and therefore the closer their associated frame rate and accuracy should be. Similar behavior is observed about parameter n. That is, *the performance of nearby configurations is correlated* (see Fig. 2). However, this correlation is not always easily captured for various reasons. Firstly, in order to acquire performance graphs like the ones shown in Fig. 2, a large dataset is required. Moreover, we consider more sophisticated reward functions that embody both optimization criteria, as well as other (user-specified) parameters.



(a) (b)

Fig. 2: Measured average CC and frame rate of our system.

C. Performance Metric

In our previous work [15], we used the mean Average Precision (mAP) and mean Average Recall (mAR) as performance metrics. These metrics should be computed offline through a large dataset of cross-validation images. Nevertheless, in an online learning setting mAP and mAR cannot be applied for several reasons. First, the performance of the selected configuration should be revealed to the online algorithm right after applying it. Moreover, the true labels of the images are not available when the system is operating.

Hence, we propose here to use the Cumulative Confidence (CC), defined as the sum of confidence values that YOLO outputs for all detected objects in an image. Unlike the mAP, this metric is available per image and favors our online setup. Fig. 2a-2b show the measured average CC and frame rate respectively, for specific values of NN size and encoding rate. We observe in this figure the same trend shown by the mAP metric in [15]. There is a logarithmic relation between the CC versus the NN size, and moreover, the encoding rate parameter q positively affects the CC for larger NN sizes, and is of marginal value when the NN size is small. These experimental results indicate that the CC captures successfully the prediction performance of the edge service but does not require labeled data, as mAP does.

III. System Model and Learning Algorithm

In this section we formulate the MAB problem and propose an online learning algorithm to solve it.

A. Multi Armed Bandit Formulation

We consider a slot-based operation for the system, where a certain number of images (depending on the system configuration) is processed per slot. For each slot t the system runs under a specific action[1] $x_t = (n_t, q_t)$, where $n_t \in \mathcal{N}, q_t \in \mathcal{Q}$ are the NN size and encoding rate values applied at slot t. Sets $\mathcal{N}$ and $\mathcal{Q}$ include the possible NN size and encoding rate values respectively, each one of size $N = |\mathcal{N}|, Q = |\mathcal{Q}|$, and the actions space is given by $\mathcal{X} = \mathcal{N} \times \mathcal{Q}$.

Our goal is to maximize the system's average CC C, while also guaranteeing that the experienced average frame rate λ is close to a desired value $\lambda^\star$. We combine the two objectives in reward function $f(x)$ defined as:

$$f(x) = C(x) - \alpha|\lambda(x) - \lambda^\star|, x \in \mathcal{X}, \tag{1}$$

[1]The terms configuration and action are used interchangeably.

where $C(x)$ and $\lambda(x)$ are the average CC and frame rate obtained using action x, while α is a balancing parameter between the two objectives. This way, we scalarize the problem and facilitate this pareto optimization by properly selecting the weight α. Observe that the reward function increases with $C(x)$ and as $\lambda(x)$ gets closer to the desired performance $\lambda^\star$[2].

As explained before, $C(x)$ and $\lambda(x)$ can be evaluated offline using a dataset (see Fig. 2), but in practice they are unknown a priori, and hence $f(x)$ is also unknown. Moreover, after using the configuration x_t, the instantaneous observed values of CC and frame rate ($C_t(x_t)$ and $\lambda_t(x_t)$) are noisy since only a small number of images is processed during one slot. Hence, we model the instantaneous observed reward associated with action x_t as:

$$y_t = C_t(x_t) - \alpha|\lambda_t(x_t) - \lambda^\star| = f(x_t) + \epsilon_t,$$

where $\epsilon_t \sim N(0, \sigma^2)$ is random Gaussian noise. Preliminary analysis of our dataset indicates this is a fairly good model since the rewards obtained under all possible actions are Gaussian-like distributed. This is displayed in Fig.3a,3b where we show the average reward $f(x)$ obtained by our dataset for each action $x \in \mathcal{X}$, and the distribution of said rewards for two specific configurations.

We denote the set of actions applied up to slot t by $\mathcal{A}_t = \{x_1, \ldots, x_t\}$, and the observed rewards as a result of those actions by $Y_t = \{y_1, \ldots, y_t\}$. Our framework operates as follows. An action x_t is applied at slot t. At the end of this slot the observation of the reward y_t is revealed to the algorithm. Both x_t and y_t are included in $\mathcal{A}_t$ and Y_t, respectively. The action for the next slot x_{t+1} is selected based on the previous experience, i.e., $\mathcal{A}_t$ and Y_t. Thus, as the algorithm gains more experience it is expected to converge to the optimal action, denoted by $x^\star = \arg\max_{x \in \mathcal{X}} f(x)$.

We define the expected regret as the difference of the total reward obtained by our algorithm if $f(x)$ was known, from the optimal reward obtained if $x^\star$ was selected at each slot t. Formally we have:

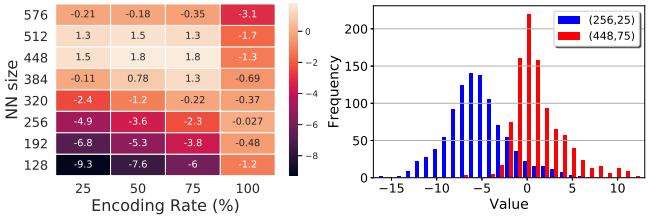
$$R_T = \sum_{t=1}^{T} f(x^\star) - \sum_{t=1}^{T} f(x_t). \tag{2}$$

Thus, *our objective is to minimize the regret*. Under this setup, we can use a variety of no-regret algorithms, i.e. yielding $\lim_{T\to\infty} R_T/T \to 0$, like the classic Upper Confidence Bounds (UCB) algorithm [14]. In our problem however, as we discussed in the previous section, there is a correlation between the different actions and their expected rewards. In the next section, we detail how our proposal exploits this structural property of the problem.

B. The GP-UCB Algorithm

Many algorithms have been proposed considering a correlation on the expected reward of similar actions [18]–[20]. A common assumption in these works is that this correlation is

[2]By using the absolute value we penalize excessive increase of the frame rate that will unnecessarily burden the system in cases where a certain range of frame rate values is acceptable.



(a) (b)

Fig. 3: (a) Correlation of action rewards across different configurations for $\lambda^\star = 25$ fps, $\alpha = 0.5$; (b) distributions of actions (256,25) and (448,75).

linear, or given by a known function. In our setting however, the reward function (1) is unknown, and the average rewards obtained by our experiments are clearly non-linear as shown in Fig. 3a. Hence, we propose to use Bayesian non-parametric strategy to model this correlation.

Specifically, we model the function $f(x)$ as a GP [21]. This way, we can compute the posterior distribution of $f(x)$ at slot t using the actions $\mathcal{A}_t$ and their observed rewards Y_t. The posterior distribution follows Gaussian distribution with mean value $\mu_t(x)$ and variance $k_t(x,x)$ given by:

$$\mu_t(x) = \boldsymbol{k}_t(x)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} Y_t, \tag{3}$$

$$k_t(x,x') = R(x,x') - \boldsymbol{k}_t(x)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(x'), \tag{4}$$

where $\boldsymbol{I}$ is the identity matrix, $R(x,x')$ is an appropriate kernel function expressing the correlation between actions x, x', $\boldsymbol{k}_t(x) = [R(x_1,x), \ldots, R(x_t,x)]^\top$, and $\boldsymbol{K}_t = [R(x_m,x_n)]$, $m,n = 1,\ldots,t$ is the positive definite kernel matrix. The above equations allow us to estimate the mean value $\mu_t(x)$, and variance $k_t(x,x)$ of $f(x)$, by only observing the noisy rewards Y_t.

For the kernel function we used the radial basis function [22]:

$$R(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\rho^2}\right), \tag{5}$$

where the distance $\|x-x'\|^2$ is calculated based on the 2-D Cartesian coordinates of the action space, where the origin is at $x_0 = (128,25) \rightarrow (0,0)$ and the action with the longest distance $x_{31} = (576,100) \rightarrow (7,3)$[3]. Observe that if $x = x'$ the kernel function is 1, and then it decreases with the distance between the actions. Note also, that for the evaluation of (3),(4) we used the observed Y_t and not the unknown f.

The idea of combining GPs with a bandit algorithm has been proposed in the seminal work [23]. To the best of our knowledge, this is the first time it is applied in configuring a real-time DL service like object recognition. GP-UCB [23] selects an action at each time slot t that has a high average reward and/or high variance, i.e. high uncertainty for its true mean reward. It uses the increasing with t parameter β_t to balance between exploration and exploitation as follows:

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \mu_t(x) + \sqrt{\beta_t k_t(x,x)} \tag{6}$$

Note that by using (3),(4) we can even estimate the average reward of previously unexplored actions, based on their distance from actions with previously observed rewards. The GP-UCB algorithm is given in Algorithm 1, and it's performance characterized by the following lemma:

[3]In our system we have a total of 32 possible actions based on the sets of possible NN sizes and encoding rates.

Algorithm 1 GP-UCB

Initialize: $x_1 \in \mathcal{X}, \alpha, \rho, \lambda^\star > 0$, s, and β_t
for $t = 1, 2, \ldots$ **do**
  Process images and obtain: $C_t(x_t), \lambda_t(x_t)$
  $y_t \leftarrow C_t(x_t) - \alpha|\lambda_t(x_t) - \lambda^\star|$
  $\mathcal{A}_t \leftarrow \{x_1, \ldots, x_t\}$
  $Y_t \leftarrow \{y_1, \ldots, y_t\}$
  for $x \in \mathcal{X}$ **do**
    $\mu_t(x) \leftarrow \boldsymbol{k}_t(x)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I}) Y_t$
    $k_t(x,x) \leftarrow R(x,x) - \boldsymbol{k}_t(x)^\top (\boldsymbol{K}_t + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}_t(x)$
  end for
  $x_{t+1} \leftarrow \arg\max_{x \in \mathcal{X}} \mu_t(x) + \sqrt{\beta_t k_t(x,x)}$
end for

Lemma 1. *Suppose $\delta \in (0,1)$ and $\beta_t = 2\log(|\mathcal{X}|t^2\pi^2/6\delta)$. The regret bound of Algorithm 1 is $\mathcal{O}(\sqrt{T\gamma_T \log|\mathcal{X}|})$ with high probability. In specific:*

$$\text{Prob}\{R_T \leq \sqrt{DT\beta_T\gamma_T}\} \geq 1-\delta,$$

where $D = 8/\log(1+\sigma^{-2})$ and $\gamma_T \leq \mathcal{O}(\log T)$ is the maximum information gain obtained bu the usage of the kernel function after T slots.

Proof. The proof follows from Theorem 1 in [23]. For our problem, the action space $\mathcal{X}$ is finite. The bound on γ_T also depends on problem specific parameters. In detail, assuming $T > |\mathcal{X}|$, γ_T is bounded by the worst case allocation of the $|\mathcal{X}|$ leading eigenvectors of $\boldsymbol{K}_{|\mathcal{X}|} = [k(x,x')], x, x' \in \mathcal{X}$. See [23] for the extended proof. ■

The bound of Lemma 1 is sublinear ($\mathcal{O}(\sqrt{T})$), and guarantees optimal no regret performance, i.e. $\lim_{T\to\infty} R_T/T \to 0$. It occurs with high probability by properly adjusting δ. As it increases, β_t also increases resulting in more frequent exploration, and hence higher regret.

C. Learning Agent Implementation for Multiple Users

The MAB framework defined in Sec. III.A can be easily extended to account for multiple users connected to the same MEC server with few modifications. Suppose that a set of $\mathcal{I}$ users are concurrently connected to the MEC server, requesting high quality object recognition, each one at a frame rate of $\lambda_i^\star, i \in \mathcal{I}$. The users share the NN deployed in the server's GPU, but can apply their individual encoding rate to the images they transmit. Thus, the system actions are represented by $x_t = (n_t, q_t^1, q_t^2, \ldots, q_t^{|\mathcal{I}|})$, where q_t^i is the encoding rate applied by user i at slot t.

We define the system's total reward function as:

$$f(x) = \sum_{i \in \mathcal{I}} \Big(C_i(x) - \alpha |\lambda_i(x) - \lambda_i^\star| \Big), \tag{7}$$

where $C_i(x)$ and $\lambda_i(x)$ are the expected CC and frame rate of user i, obtained using action x. Note that $\lambda_i(x)$ depend not only on the NN size and user i's encoding rate, but also on the encoding rates of all users, since they share the wireless channel. As with the single user scenario, the server observes the instantaneous CC and frame rate of each user $(C_{i,t}(x_t), \lambda_{i,t}(x_t))$, and evaluates the total system reward:

$$y_t = \sum_{i \in \mathcal{I}} \Big(C_{i,t}(x_t) - \alpha |\lambda_{i,t}(x_t) - \lambda_i^\star| \Big). \tag{8}$$

Algorithm 1 can be executed for multiple users by only applying the above adjustments. Note that the increase in the dimensionality of the action space after adding more users to the system, does not impact the computational load of dot products and matrix inversions in (3),(4), since $\boldsymbol{k}_t, \boldsymbol{K}_t, Y_t$ increase with t. The computational complexity is only increased due to the larger size of the action space.

However, there are several practical considerations. GP-UCB requires more processing time than simpler approaches like e.g. UCB. The delay can be mitigated by simply restarting the algorithm, i.e. tossing the current observations, or considering the most recent ones, as proposed in [24]. This way the size of the involved vectors and matrices in (3),(4) does not continuously increase with t. Another option is pruning parts of the action space by limiting it to a number of competitive (high reward) actions [9], [25]. This is even more important if there are multiple users in the system since each added user scales the action space by $|\mathcal{Q}|$.

IV. Performance Evaluation

In the following, we evaluate the performance of our solution in the deployed MEC object recognition system.

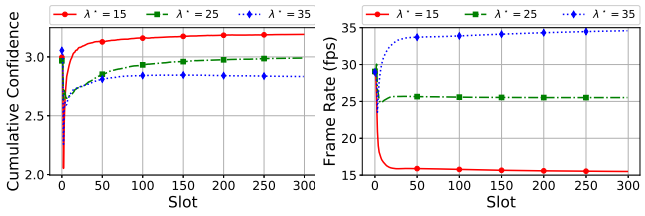
A. Experimental Setup

We evaluated the object recognition system on our testbed comprised of a 3.7 GHz Core i7 PC, with 32 GB of RAM and a GeForce RTX 2080 Ti GPU as the server; and a Google Pixel 2 as the mobile device. The wireless connection between them was established using 802.11ac with the ASUS RT-AC86U router on the 5 GHz band.

Firstly, we gathered per image CC and latency measurements for each possible system action, using a pre-trained model on the COCO dataset [26]. In detail we defined $\mathcal{N} = \{128, 192, 256, 320, 384, 448, 512, 576\}$, and $\mathcal{Q} = \{25, 50, 75, 100\}$. We used the above measurements to evaluate the performance of Algorithm 1, by randomly sampling 20 measurements, i.e. images, for each time slot. The results presented have been averaged from 100 runs.

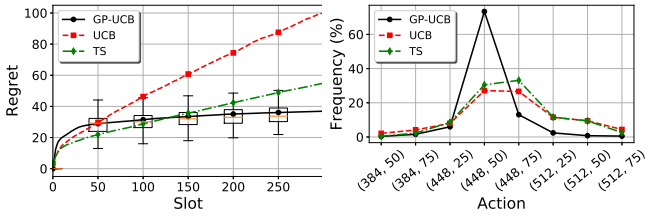
B. Experimental Evaluation

We evaluate the performance of GP-UCB with respect to the performance metrics, i.e. CC and frame rate, as well as the optimality measure, i.e. the regret. We compare the latter, with the regret achieved by standard algorithms



(a) (b)

Fig. 4: Running average of Cumulative Confidence and frame rate of GP-UCB for different values of the desired rate $\lambda^\star$.



(a) (b)

Fig. 5: (a) Regret comparison under $\lambda^\star = 25$ fps. (b) Action selection frequencies for the different algorithms.

used in the MAB literature like UCB [14] and Thompson Sampling (TS) [27]. In brief, UCB selects actions based on a combination of expected reward and selection frequency of each action. TS randomly samples a (gaussian for our problem) distribution for each action, based on the observed rewards, and selects the one with the highest outcome.

Fig. 4a-4b display the running average of CC and frame rate (defined as $1/t \sum_{\tau=1}^{t} C_\tau(x_\tau)$ and $1/t \sum_{\tau=1}^{t} \lambda_\tau(x_\tau)$ respectively) achieved by Algorithm 1 for different values of the target frame rate $\lambda^\star$. The efficiency of the algorithm is increased as the CC increases and the frame rate is close to the rate $\lambda^\star$. As expected, for higher $\lambda^\star$, the CC gets smaller to allow a (higher) frame rate, closer to the desired $\lambda^\star$. Moreover, we observe that after the initial slots where the algorithm explores the correlation between actions, the CC increases until it reaches a stable average value, while the frame rate continues to approach $\lambda^\star$, as the algorithm keeps selecting high reward actions more frequently.

We depict the cumulative regret of GP-UCB, UCB and TS algorithms in Fig. 5a. Compared to UCB and TS, GP-UCB shows at first higher regret, since it needs to explore various areas of the action space to reduce the initially high uncertainty. However, it quickly manages to locate the high reward actions and after 150 slots its regret is lower than both UCB and TS. The latter, also outperforms UCB, since it takes advantage of the Gaussian distribution of the rewards, while UCB tends to explore low reward actions that have not been selected too often before.

The frequency of selected actions is displayed in Fig. 5b, only for the actions selected for more than 1% of the time. Note that actions (448,50) and (448,75) are the ones yielding the highest rewards (see Fig. 3a). Observe that one of these actions is selected by GP-UCB for about 86% of the time

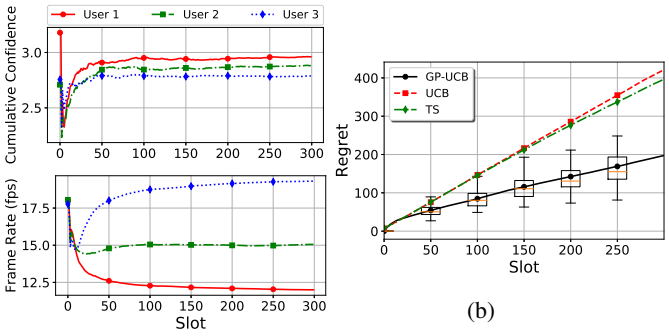


Fig. 6: (a) CC and Frame rate performance of Algorithm 1, and (b) regret comparison with UCB and TS, for 3 users.

while for UCB and TS this percentage is only 54% and 64% respectively. Instead they apply lower reward actions more frequently, e.g. (512,25), resulting in higher long term regret.

C. Multi-User Evaluation

We generated a new dataset for 3 users, using the measurements obtained for a single mobile device and adding random transmission and YOLO processing delays to emulate medium access and GPU resource contention. The CC performance of each device, naturally remains unaffected by the number of users in the system.

In Fig.6 we present the performance of a multi-user scenario. In specific, we consider a set of 3 users with target frame rates: $\lambda_1^\star = 10, \lambda_2^\star = 15, \lambda_3^\star = 25$. In Fig.6a (upper) we observe fluctuations in the CC, as the algorithm frequently tries different NN sizes, towards adapting to the varying needs of the users. Regarding the frame rate in Fig. 6a (lower), we observe a delayed stabilization compared to the single user case, as well as a substantial divergence from the target rate in the extreme cases (user 1 achieves 12 instead of 10 fps, user 3 achieves 20 instead of 25 fps) as a result of the coupling of the NN size between the users.

Regarding the regret displayed in Fig. 6b, we observe a similar trend with Fig. 5a with the difference that the regret does not flatten after 300 slots. This indicates that all algorithms still have to explore the vast action space (512 actions from 32 in the single user case.) UCB and TS perform about the same but GP-UCB manages to maintain a much lower regret increase rate, indicating it selects actions, much closer to the optimal than its competitors.

V. Conclusions

In this paper we designed a solution for configuring the NN size and encoding rate of an example edge-assisted object recognition system. We exploit its structural properties to formulate and solve, using an online algorithm, the problem of jointly optimizing the system accuracy and frame rate. Our data-driven solution makes no assumptions on the form of the objective function, and can easily be extended to account for multiple users that share wireless and edge computing resources. The results demonstrate the dominance of our algorithm over other approaches, as it is able to learn the optimal configurations much faster.

References

[1] Y. Han *et al.*, "Convergence of edge computing and deep learning: A comprehensive survey," *CoRR*, vol. abs/1907.08349, 2019.

[2] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[3] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[4] X. Ran *et al.*, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. of IEEE INFOCOM*, 2018.

[5] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. of ACM MobiCom*, 2019.

[6] T. Y.-H. Chen *et al.*, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proc. of ACM SenSys*, 2015.

[7] P. Jain, J. Manweiler, and R. Roy Choudhury, "Overlay: Practical mobile augmented reality," in *Proc. of ACM MobiSys*, 2015.

[8] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. of USENIX NSDI*, 2017.

[9] J. Jiang *et al.*, "Chameleon: Scalable adaptation of video analytics," in *Proc. of ACM SIGCOMM*, 2018.

[10] Y. Wu *et al.*, "A learning-based expected best offloading strategy in wireless edge networks," in *Proc. of IEEE GLOBECOM*, 2019.

[11] F. Wang *et al.*, "Deepcast: Towards personalized qoe for edge-assisted crowdcast with deep reinforcement learning," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[12] C. Wang *et al.*, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. of IEEE INFOCOM*, 2020.

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

[14] 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.

[15] A. Galanopoulos, V. Valls, G. Iosifidis, and D. J. Leith, "Measurement-driven Analysis of an Edge-Assisted Object Recognition System," *to appear in Proc. of IEEE ICC, 2020, available at: arXiv:2003.03584*.

[16] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of IEEE CVPR*, 2016.

[17] W. Wang *et al.*, "A survey on applications of model-free strategy learning in cognitive wireless networks," *IEEE Comm. Surv. & Tut.*, vol. 18, no. 3, pp. 1717–1757, 2016.

[18] S. Filippi, O. Cappe, A. Garivier, and C. Szepesvári, "Parametric bandits: The generalized linear case," in *Advances in Neural Information Processing Systems*, 2010.

[19] Y. Deshpande and A. Montanari, "Linear bandits in high dimension and recommendation systems," in *Proc. of Conference on Communication, Control, and Computing (Allerton)*, 2012.

[20] Y. Russac, O. Cappé, and A. Garivier, "Algorithms for non-stationary generalized linear bandits," *arXiv preprint arXiv:2003.10113*, 2020.

[21] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[22] C. M. Wu *et al.*, "Mapping the unknown: The spatially correlated multi-armed bandit," *bioRxiv*, 2017. [Online]. Available: https://www.biorxiv.org/content/early/2017/04/28/106286

[23] 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.

[24] I. Bogunovic, J. Scarlett, and V. Cevher, "Time-varying gaussian process bandit optimization," in *Proc. of International Conference on Artificial Intelligence and Statistics*, 2016.

[25] S. Gupta, G. Joshi, and O. Yagan, "Exploiting correlation in finite-armed structured bandits," *CoRR*, vol. abs/1810.08164, 2018. [Online]. Available: http://arxiv.org/abs/1810.08164

[26] T. Lin *et al.*, "Microsoft COCO: common objects in context," *arXiv*, vol. abs/1405.0312, 2014.

[27] H. Junya and T. Akimichi, "Optimality of thompson sampling for gaussian bandits depends on priors," *arXiv preprint arXiv:1311.1894*, 2013.