Li, Xuebing; Cho, Byung; Xiao, Yu

Balancing Latency and Accuracy on Deep Video Analytics at the Edge

Published in:
2022 IEEE 19th Annual Consumer Communications & Networking Conference

DOI:
10.1109/CCNC49033.2022.9700636

Published: 01/02/2022

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Li, X., Cho, B., & Xiao, Y. (2022). Balancing Latency and Accuracy on Deep Video Analytics at the Edge. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference* (pp. 299-306). (IEEE Consumer Communications and Networking Conference). IEEE. https://doi.org/10.1109/CCNC49033.2022.9700636

Balancing Latency and Accuracy on Deep Video Analytics at the Edge

Xuebing Li, Byungjin Cho, and Yu Xiao
Department of Communications and Networking, Aalto University, Espoo, Finland
Email:{Xuebing.Li, Byungjin.Cho, Yu.Xiao}@aalto.fi

***Abstract*—Real-time deep video analytic at the edge is an enabling technology for emerging applications, such as vulnerable road user detection for autonomous driving, which requires highly accurate results of model inference within a low latency. In this paper, we investigate the accuracy-latency trade-off in the design and implementation of real-time deep video analytic at the edge. Without loss of generality, we select the widely used YOLO-based object detection and WebRTC-based video streaming for case study. Here, the latency consists of both networking latency caused by video streaming and the processing latency for video encoding/decoding and model inference. We conduct extensive measurements to figure out how the dynamically changing settings of video streaming affect the achieved latency, the quality of video, and further the accuracy of model inference. Based on the findings, we propose a mechanism for adapting video streaming settings (i.e. bitrate, resolution) online to optimize the accuracy of video analytic within latency constraints. The mechanism has proved, through a simulated setup, to be efficient in searching the optimal settings.**

***Index Terms*—Video streaming, deep learning inference, object detection**

I. Introduction

The proliferation of deep learning and visual sensors has populated deep video analytic. For mobile users, offloading deep video analytic from mobile devices to the edge cloud has been commonly applied for reducing the processing latency and the energy consumption of mobile devices [1], [2]. Since the accuracy of video analytic such as object detection is affected by the quality of the video, it is desired to stream video with high resolution and bitrate, which, however, would incur high transmission and processing latency and consume more network bandwidths, invalidating the overall latency constraint for latency-intensive applications [3]–[5]. To address this issue, it is essential to balance the trade-off between the latency and the accuracy in the design of deep video analytics at the edge.

Existing work [1] balances the tradeoff with a data-driven method: the control variables' (e.g., resolution's and bitrate's) effects on the performance metrics (e.g., latency and accuracy) are modeled beforehand and the optimization problem is solved based on the inferred model. However, the data-driven method is not generic as it is tightly coupled with a concrete setup, in terms of software, hardware, and network. In this paper, we propose an online search algorithm that searches for the best setting of control variables to fulfill the optimization target without prior knowledge of the system in execution. The proposed algorithm works by trying with a series of settings in control variables, based on the observation that both performance metrics are monotonic to the control variables, to find out the one that maximizes the performance, i.e., having the highest inference latency while guaranteeing the overall latency smaller than a threshold.

To prove the effectiveness of the proposed algorithm, we set up a deep video analytics testbed for case study. Web Real-Time Communication (WebRTC) [6] and YOLO [7] are chosen for video streaming and deep learning inference, respectively. By experimenting with all settings of control variables, we formulate each control variable's effect on the performance metrics in the given setup. Then, the formulation is fed into a simulator, which runs the proposed algorithm for verification. The evaluation result shows that the online search algorithm is efficient in finding out the optimal setting of control variables.

The contributions of this paper are summarized below:

- Deep analysis of the trade-off between latency and accuracy and the impact from different control variables through extensive measurement of real-time edge analytic of video streams.
- An efficient online search algorithm that maximizes the inference accuracy within latency constraints. The proposed algorithm implemented in WebRTC proves to achieve higher accuracy with lower latency, without prior knowledge on the software/hardware setup and network condition.

The rest of this paper is organized as follows: Sec. II describes background knowledge, control variables, and performance metrics that we focus on in this paper. In Sec. III, we formulate the optimization problem and present our solution to it. In Sec. IV, we present a case study with a concrete deep video analytics application and, based on the measurement result from it, we prove the effectiveness of the proposed algorithm via simulation. Finally, we discuss related topics in Sec. V and conclude our work in Sec. VI.

II. Background, Metrics and Control variables

In this section, we explain the background of video streaming and deep learning inference with the interest of performance metrics and control variables.

A. Background

Fig. 1 gives an overview of typical architecture of edge-assisted deep video analytic. A general video streaming pipeline is composed of three stages: encoding, streaming, and decoding. The User Equipment (UE) collects video frames from its equipped webcam, encodes the frame via a video codec, and sends it out over the network. On reception of a frame, a mobile edge computing (MEC) node decodes it into raw RGB format, which is the input of the inference program.

Video streaming. Due to lossy compression implemented in the encoding/decoding process in video codecs, the amount of data to be transmitted decreases at the expense of extra processing delays at both the sender and receiver sides. Although the encoding and decoding processes take more time than the transmission process under a network, e.g., 5G [8], they are indispensable because of the efficiency in reducing the size of data sent over the network. In this paper, we propose a generic online search algorithm that works with any video streaming platform. Limited by the workload, we implement the proposed algorithm in WebRTC, which is one of the most popular video streaming platforms as its wide support by the browsers and mobile operation systems [9], for evaluation. The default setting of WebRTC is used: we use VP8 for encoding and decoding a frame, Real-time Transport Protocol (RTP) for data transmission, and google congestion control (GCC) [6] for congestion control. Besides, we modify the source code of WebRTC to control parameters, i.e., resolution and bitrate and to collect performance metrics, i.e., encoding/decoding latency, frame transmission latency.

Deep Learning Inference. Benefiting from the advance in computer vision with the help of deep learning models such as Deep Neural Networks (DNN), state-of-the-art inference algorithms outperform human eyes for visual recognition problems [10] and are already widely used in autonomous driving, such as vehicle and pedestrian detection [11] and traffic lane detection [12]. All of them motivate deep video analytics for improving driving safety. In this paper, we choose one of the state-of-the-art object detection algorithm YOLOv5 [7] as case study to verify the effectiveness of the proposed algorithm. We use two pre-trained DNN models provided by YOLOv5, which detects, but are not limited to, pedestrians, vehicles, and bikes. YOLOv5 takes a raw RGB image, which is the output of video streaming, as input and outputs several bounding boxes of the detected objects. Internally, YOLOv5 uses threshold filtering and non-maximum suppression to remove low-confidence bounding boxes and repeated bounding boxes of the same object, configurable via confidence threshold and Intersection over Union (IoU) threshold, respectively.

B. Performance metrics

We consider two important performance metrics in the process of deep video analytic, namely, latency and accuracy.

Latency. The latency is calculated as the time spent between capturing a video frame on the UE and generating the inference output on the MEC. As shown in Fig. 1, the overall latency is composed with the following latencies:

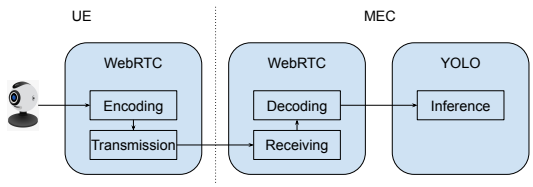


Fig. 1: Architecture of deep video analytics at the edge

- Encoding / Decoding latency. The time spent on converting the raw RGB image into a smaller size and vice versa, which is CPU-intensive.
- Frame transmission latency. The time spent on sending and receiving all of the packets of a frame, which is network-intensive.
- Inference latency. The time spent on processing a frame by the inference program, which is GPU-intensive.

Accuracy. The inference accuracy is measured by the mean average precision (mAP) widely used in YOLO-related works [7], [13]. Basically, the more objects are detected and the closer the detection bounding box is to the ground truth, the higher mAP is. The average of the frame mAPs gives the video's mAP. The ground truth is provided by the Waymo dataset [14].

C. Control variables

There exist a number of variables that affect the performance metrics in deep video analytics. How to tune these parameters to get the best performance motivates our work.

Resolution. Resolution defines how many pixels are used for representing a frame. In this case, a higher resolution inevitably results in more data being stored in memory and being processed in CPU/GPU. For video streaming, as bitrate regulates the compressed size, given the same bitrate, a higher resolution means that a higher compression ratio is required when encoding and decoding a frame. For deep learning inference, state-of-the-art DNNs use a convolutional neural network (CNN) to traverse over all of the pixels in the input frame [7]. So, a higher resolution means more computation is required for processing a frame. So, both encoding/decoding latency and inference latency are, at least, non-decreasing to the resolution. On the other hand, more pixels in a frame means more content is stored, giving DNN more opportunity to have a good inference accuracy.

Bitrate. Bitrate regulates the amount of data encoded for a unit of time. When the frame rate is constant, bitrate actually controls the size of the encoded frame. Modern video codec is based on frequency transforms [15], where spatial RGB data is transformed into frequency domain and high-frequency information are lost for compression. Visually, high-frequency information represents details in an image and losing it makes the image look blurry. So, a lower bitrate makes a frame harder to recognise, even for human, and reduces the inference accuracy. On the other hand, because less frequency transforms are required, a lower bitrate also means a smaller processing delay for both encoding and decoding.

Variables	Description	Metrics	Description
b	Streaming bitrate	l_e	Encoding latency
r	Frame resolution	l_d	Decoding latency
m	DNN model	l_t	Frame transmission latency
Γ	Delay limit	l_i	Inference latency
Ω	Frame rate	a	Object detection accuracy

TABLE I: Control variables and metrics. For differentiation, r denotes the frame resolution used in video streaming and r' denotes the frame resolution used in inference.

DNN model. Modern DNN models are composed of hundreds or even thousands of layers [16]. Each layer requires a step of computation in the prediction phase. So, more layers mean that more time is spent on calculating the result, thus higher inference latency. The benefit of deeper layers is that the DNN can construct a more complex network, which helps in understanding the context of an image. So, a heavier DNN model usually means higher inference accuracy.

One may find that all of the control variables have a monotonic effect on the performance metrics: it is either non-increasing or non-decreasing. However, the effect is not easy to model. Firstly, the monotonic effect is not constant. For example, increasing the bitrate from 1 Mbps to 2 Mbps may increase the inference accuracy a lot but the effect may be not observable if the bitrate is increased from 9 Mbps to 10 Mbps, although the increment is the same for these two cases. Secondly, a real-world application is so complex that many other factors affects one control variable's affect on the performance metrics. For example, down-scaling the resolution before inference may reduce the processing latency a log on a moderate machine, but not on a powerful machine. So, how to dynamically adjust the parameters on different conditions to achieve the optimal point between latency and accuracy is a challenging problem.

III. Our Algorithmic Problem

This section describes the problem, our optimization framework, and the algorithms to solve this optimization problem.

A. Problem

We aim to create a general optimization framework, where any control variables' effects on the performance metrics are not well formulated beforehand, that helps edge-based deep video analytics achieve the optimal balance between latency and accuracy by tuning the control variables, including i) the frame resolution r for video streaming, ii) both the DNN model m and the frame resolution r' for deep learning inference, and iii) streaming bitrate b for both video streaming and deep learning inference. The control variables' effects on the performance metrics are shown in Fig. 2. We try to solve an optimization problem at the beginning of each frame's execution to determine the configuration parameters

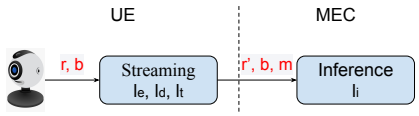


Fig. 2: Control variables' impact on data flow

for deep learning inference. The optimization problem can be formulated as below:

$$\max_{r',b,m} \quad a(r',b,m) \tag{1a}$$

$$\text{s.t.} \quad l_e(r,b) + l_d(r,b) + l_t(b) + l_i(r',m) \leq \Gamma \tag{1b}$$

$$b_{min} \leq b \leq b_{max} \tag{1c}$$

$$r_{min} \leq r \leq r_{max} \tag{1d}$$

$$r_{min} \leq r' \leq r \tag{1e}$$

where the objective (1a) is to maximize the inference accuracy, $a(r',b,m)$, determined by deep learning model, m and quality of frame content controlled by high streaming bitrate, b and high frame resolution, r', i.e., better frame content with high complexity interference model would achieve higher inference precision but at the cost of delay which may occur in different phases of offloading operations [1]. The constraint function in (1b), referring to the overall latency of processing a single frame, could not exceed a pre-defined threshold Γ, as a common regulation in service-level agreements (SLAs) [17]. The overall latency consists of, as in Sec II-B, the following components: i) encoding, $l_e(r,b)$ ii) frame transmission, $l_t(b)$, iii) decoding, $l_d(r,b)$, and iv) inference, $l_i(r',m)$, each of which is itself a function of some or all variables, r, b, r', and m. The effects of the parameters on the components in the latency are described in Sec II-C. While the constraints in (1c), (1d) and (1e) allow the application to define minimum and maximum allowable values of bit rate, b, and frame resolutions, r and r', respectively, the constraint (1e) indicates that the frame resolution for deep learning inference, r', cannot be chosen to exceed the one for video streaming, r, from which the deep learning inference may further reduce the resolution to save more time on image inference. All variables involved in the problem (1) are denoted as shown in Table I.

Although the control variables' effects on each performance metric are intuitive, as what we described in Section II-C, there is no easy way to formulate their relationship mathematically because, as what we will show in Section IV-B, the formulation is highly coupled with the system implementation. So, it is in practice impossible to solve the optimization problem directly. A brute-force solution is to exhaust all pairs of control variables to find out the optimal solution. However, it is neither practical nor efficient due to high complexity, especially in an online environment.

B. Solution

To solve this problem, we propose an online search algorithm to find the best settings of parameters. The proposed

Algorithm 1 Running on the MEC side

```
1: function ParameterTuneServer
2:     while frame ← ReceiveFrame() do
3:         r' = frame.r
4:         while l_e + l_d + l_t + l_i(r', m) > Γ and frame.r > r_min do
5:             Decrease(r')
6:         end while
7:         ProcessFrame(frame)
8:         SendFeedback(r')
9:     end while
10: end function
```

Algorithm 2 Running on the UE side

```
1: function ParameterTuneClient
2:     b ← b_min
3:     r ← r_max
4:     a ← 0
5:     while b ≤ b_max and r ≥ r_min do
6:         frame ← NextAvailableFrame()
7:         feedback ← StreamFrame(frame, r, b)
8:         if feedback.r' < r then
9:             Decrease(r)
10:        else
11:            Increase(b)
12:            if feedback.a > a then
13:                a ← feedback.a
14:                setting_opt ← (r, b)
15:            end if
16:        end if
17:    end while
18:    return setting_opt
19: end function
```

algorithm is composed of two parts, which run on the MEC and the UE, separately.

MEC side algorithm. The purpose of Algorithm 1 is to guarantee that the overall latency of processing a frame is within the latency constraint. Note that the output of video streaming is the baseline of the input of deep learning inference. So, in a real application, if a frame already spends lots of time on streaming, the inference algorithm can further reduce the resolution to save time on processing. Upon receiving and decoding a frame, the algorithm calculates the time left for further operation by deducing the latency spent in the previous stages from the latency limit (line 4). Then, the algorithm chooses the highest allowable setting in the resolution that can be processed within the left time (line 5). After processing (line 7), the receiver responds with the actual resolution in use (line 8) to the sender as feedback. The feedback indicates whether the current setting causes too much latency: if $r' < r$, the sender can reduce the streaming resolution to r' to get lower latency but does no harm to the inference latency, which motivates the design of Algorithm 2.

UE side algorithm. The purpose of Algorithm 2 is to search for the setting of control variables that achieves the optimization target. Initially, the streaming bitrate is set to the minimal value and the frame resolution is set to the maximum value (lines 2-3). On the feedback of a sent frame (line 7), one may check if the inference program has reduced frame resolution (line 8) to save time. If so, it means that the current setting consumes too much time in streaming and the resolution should be decreased (line 9). Otherwise (line 10), it means that more time can be spent on streaming to improve frame quality and achieve higher inference accuracy. So, we increase the bitrate for further probing (line 11). With different sets of control variables, we record their corresponding performance metrics and the optimal setting is found when the bitrate reaches the maximum value, or the resolution reaches the minimum value (lines 12-14).

How it works. The output of the algorithm is guaranteed to be the optimal one because, for each given resolution, the highest bitrate without violating the latency constraint is examined. On line 11 in Algorithm 2, whenever the current setting fulfills the latency requirement, it tries to increase the bitrate until the latency constraint is no longer kept to maximize the inference accuracy. Then, a lower resolution is used for further examination. In this way, we get the optimal setting of each resolution, and the best one among them is actually the optimal setting for the current system.

In summary, with the help of receiver side feedback, the sender goes through a searching path and picks the optimal setting where the inference accuracy is maximized while the overall latency is smaller than a threshold. During this period, the overall latency is constantly below the threshold as the receiver proactively controls the inference latency by downscaling input frames. Note that the bitrate increases from the lowest value to the highest value in Algorithm 2 just like congestion control algorithms raising sending rate during slow-start [18]. The execution of the proposed algorithms can be bounded with the transport layer's bandwidth probing phase, which happens just after connection establishment. When a transport layer bandwidth is finally found by the congestion control, Algorithm 2 also finds the best setting of the control variables that optimizes the performance metrics.

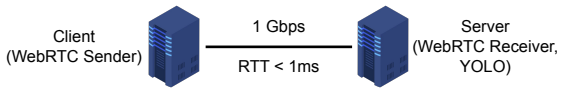


Fig. 3: Experiment setup.

IV. Experiments

We conduct real-world measurements with the setup described in Section IV, and estimate the formula of $a(r', b, m)$, $l_i(r', m)$, $l_e(r, b)$, and $l_d(r, b)$ via empirical measurements. The estimated formulas are further used to evaluate the effectiveness of the proposed algorithm in Section IV-C.

A. Experimental setup

The experiment setup is shown in Fig. 3. All of the machines run 64-bit Ubuntu 20.04 LTS. The client is equipped with an Intel Xeon E3-1230 CPU, 16 GB memory, and a Quadro K2200 GPU. The server has two different specifications: one is the same as for the client machine, denoted as MEC1, and the other one is equipped with an Intel Xeon E3-1230 CPU, 16 GB memory, and a Quadro P5000 GPU, denoted as MEC2. They are connected to the same Ethernet switch with a capacity of 1 Gbps. The RTT between each two of them is smaller than 1 ms. The setup guarantees that the network is not the bottleneck in the experiment.

One WebRTC instance runs on the server to receive video frames and store them in shared memory. In a different process, the YOLO program reads each frame from the shared memory and performs object detection. Another WebRTC instance runs on the client to stream frames to the server at 10 frames per second, e.g., $\Omega = 10$, as the settings in [19]. The video source is fed from the Waymo open dataset.

In video streaming, we configure the bitrate by hard-coding the corresponding parameter when invoking the encoder. The video clips are adjusted to different scales to change the resolution, from 320p to 1280p [1]. To collect WebRTC's latency metrics, we modify its source code to log a timestamp at each stage of execution and use the deduction of timestamps as the measured latency. In object detection, it reads from WebRTC receiver's output and uses pre-trained models for object detection.

B. Measuring control variables' effects on performance metrics

Encoding/decoding latency. Frame encoding/decoding latency is the most computationally intensive operation in video streaming, taking up most of the time when sending and receiving frames.

As shown in Fig. 4, both encoding and decoding latency values increase w.r.t bitrate and resolution, in general. For low resolution, it is observed that the codec latency keeps unchanged even for higher bitrates. The reason is that the codec has a built-in threshold for each resolution and it has already contained all necessary information for the given resolution. Thus, when the setting of bitrate exceeds this value, the size of the encoded frame does not change anymore as in Fig 5. The threshold is around 2 Mbps and 5.5 Mbps for 320p and 480p, respectively. For higher resolution, the threshold should also be higher but is not in our test case range.

One may also see in Fig. 4 that the encoding latency is about 2 times greater than the decoding one, which is because decoding algorithms is simpler than encoding ones in general. Such higher-level encoding complexity is unfavorable for deep video analytic, especially in task offloading scenarios where a service requester has limited computing capacity.

Inference latency. It is known that inference latency of a DNN-based system is typically affected by two factors: operation counts and hardware performance [20]. In our experiment, the operation count is configurable by tuning the frame resolution, r and the DNN model, m.

Fig. 6 shows the inference latency over different resolutions on MEC1 and MEC2 when using YOLOv5s and YOLOv5x. The inference latency on MEC2 is lower than the one on MEC1, and YOLOv5s results in much lower latency than YOLOv5. These observations are based on the fact that i) the computational capability of GPU on MEC2 is much stronger than the one in MEC1, and ii) YOLOv5x has a deeper neural network than YOLOv5s [2]. As shown in Fig. 6 (a), while, for both DNN models on MEC1, the inference latency increases w.r.t resolution in general, YOLOv5x is more sensitive to the change in resolution. YOLOv5s requires around 23.2 ms more for processing 1920p frames than 320p frames. With YOLOv5x, the additional delay increases to 137.5 ms. This is because both higher resolution and more complex DNN model increase the number of operations to be executed in the GPU, resulting in an increased processing delay.

Such resolution-dependent bias can be still captured on MEC2, but there is a slightly different behavior concerning resolution as shown in Fig. 6 (b). With YOLOv5s, the impact of resolution is minimal, resulting in more or less 10 ms of the inference latency for all available resolutions. This is because the computational capability of GPU on MEC2 is powerful enough such that most computing operations are executed in parallel, which may not increase a processing delay. Thus, base on this, the CPU-intensive tasks, e.g., memory copy work, become a dominant factor for the inference delay. Similarly, the inference latency does not change much at lower resolutions with YOLOv5x as it does not have enough computing operations to saturate the GPU's parallel computation capability. But, at higher resolutions, the number of computing operations exceeds the GPU's parallel capability, making the inference latency increase linearly as the number of operations.

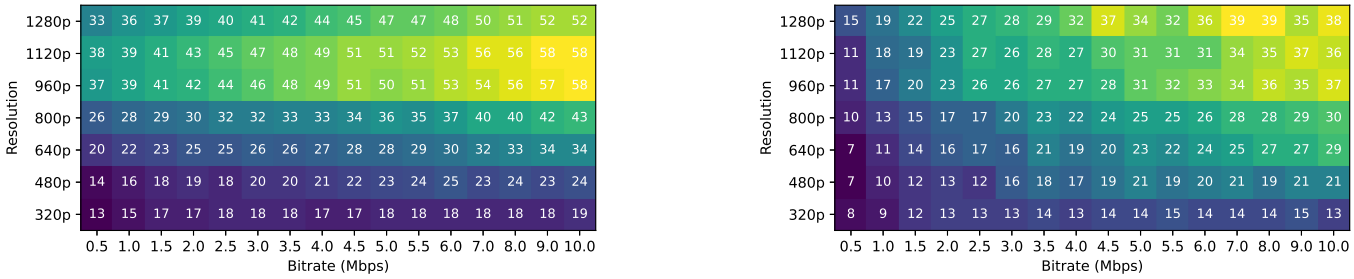
Comparing MEC1 and MEC2, both of them prove that the inference latency (l_i) is non-decreasing to the resolution (r). However, their relationship varies a lot on different hardware platforms, making it impossible to have a formula describing $l_i(r, m)$ for all of the cases.

Network transmission latency. As measured in [8], we set RTT to 4ms and bandwidth to 300 Mbps (uplink), respectively. So, when the bitrate is set to 8 Mbps, the frame transmission latency is estimated as 6.7 ms. While this estimation may not be always applicable for complex real-world networks, we focus on measurement-based analysis of processing behaviors primarily on two ends, mobile user and edge computing node assumed to be connected in a stable 5G network where handling the dynamics of network performance in a wireless network is not considered as in [21], left for future work.

Inference accuracy. The accuracy of computer vision tasks on video streams is determined by the effectiveness of two domains: data input and data processing, i.e., high-quality video

[1]The aspect ratio is 3:2, so 1920p means 1920 x 1280 pixels.

[2]The number of neural network layers with YOLOv5x and YOLOv5s are 607 and 283, respectively.



(a) Encoding latency (ms)

(b) Decoding latency (ms)

Fig. 4: Resolution and bitrate's impact on encoding latency and decoding latency.

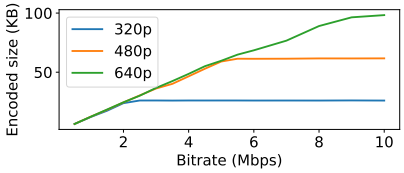
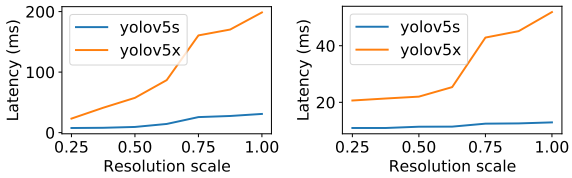


Fig. 5: Relationship between bitrate and a frame's encoded size.



(a) MEC1, moderate spec

(b) MEC2, high spec

Fig. 6: Inference latency over different resolutions.

input allows for extracting explicit attributes, and advanced inference algorithm allows for learning the faithful characteristics of the data input. With the increased accessibility to large computing capability, the qualities of the data input and data processing end up influencing the inference accuracy in various vision applications [22]. In the following, we study how the respective domains, i.e., quality of input image and deep inference algorithm, affect the inference accuracy. Mean average precision (mAP) is used as the indicator for object detection accuracy.

Fig. 7 shows the object detection accuracy w.r.t bitrate and resolution. In general, the larger the bitrate and resolution are, the higher the accuracy is. Comparing the horizontal lines, the inference accuracy increases rapidly on low bitrates but keeps stable when the bitrate reaches a certain value. It is because a large amount of visual information is lost when the video is compressed into a low bitrate, causing performance downgrading on the inference accuracy. When the bitrate is high enough, increasing bitrate does not add much detail to the decoded frame, so its effect on the inference accuracy is limited. To further analyze how bitrate affects inference accuracy at a given resolution, we compare the highest achieved accuracy

r'	480p	720p	960p	1200p	1440p	1680p	1920p
a	0.304	0.409	0.497	0.523	0.558	0.574	0.591
a'	0.288	0.390	0.479	0.509	0.536	0.554	0.571
b_{95}	1.0	1.5	1.5	1.5	2.5	2.0	2.5
b_{98}	1.5	3.0	3.0	3.5	3.5	4.0	4.0

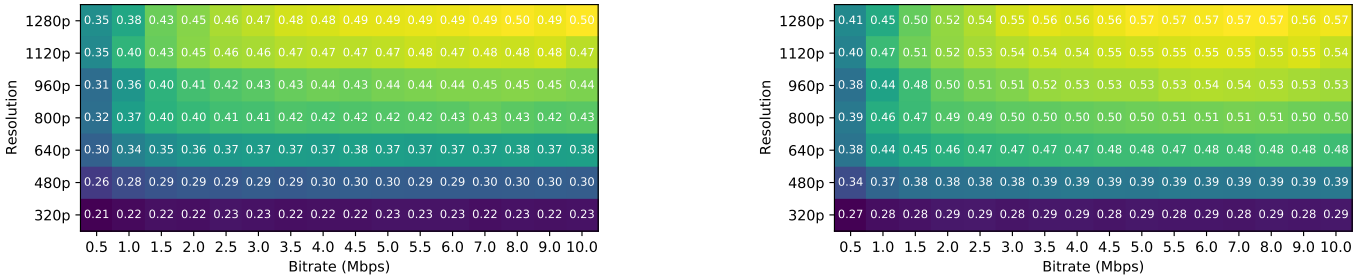
TABLE II: The inference accuracy of the original dataset as a baseline and the best accuracy after codec, a and a' (mAP), and the minimal bitrates for achieving 95% and 98% of the best accuracy, b_{95} and b_{98} [Mbps], w.r.t resolution, r'.

after video streaming with the original video (denoted as baseline accuracy) in Table II. It shows that the inference accuracy on streamed video frames is worse than processing the original video directly, with a loss in mAP of about 0.02 in all of the cases. We attribute the reason to the fact that video coding is lossy and high-frequency features in a frame are dropped, no matter how large the bitrate is set. On the other hand, we show that a relatively low bitrate is sufficient for getting a high inference accuracy. For instance, in 1920p, 4 Mbps is sufficient for achieving an accuracy no worse than 98% of the best value. The requirement on bitrate is even as small as 2.5 Mbps if we are satisfied with an accuracy of 95% of the best value. Comparing the vertical lines in Fig. 7, it is observed that it is always beneficial on using a higher resolution at a given bitrate, regarding inference accuracy. The reason is that down-scaling algorithms are usually not as efficient as video codecs on retaining high-priority information when compressing a frame.

C. Performance evaluation

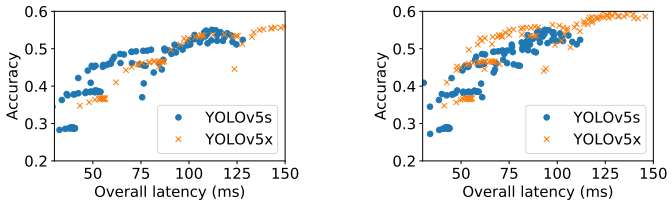
In this subsection, we firstly analyse the tradeoff between latency and accuracy based on the above mentioned measurement results and then evaluate the proposed algorithm based on the measurement data.

Latency vs accuracy. Fig. 8 shows the correlation between latency and accuracy by experimenting with different settings of control variables. As mentioned above, the resolution ranges from 320p to 1280p and the bitrate ranges from 0.5 Mbps to 10 Mbps. The latency threshold (Γ) is set as 100 ms [2]. Comparing Fig. 8a and Fig. 8b, MEC2 is, in general, of better performance, i.e., higher accuracy and lower latency, as its



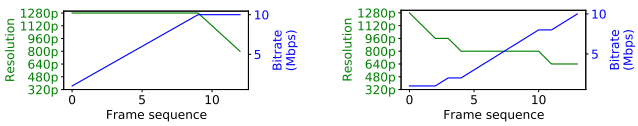
(a) YOLOv5s (b) YOLOv5x

Fig. 7: Inference accuracy over different resolution and bitrate. Using different YOLO models.



(a) MEC1 (b) MEC2

Fig. 8: Latency versus Accuracy over various settings of video streaming variables.



(a) YOLOv5s (b) YOLOv5x

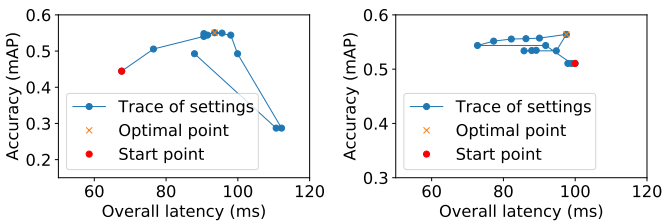
Fig. 9: Timeline of searching for the optimal setting

hardware is more powerful. And, one both machines, there exist lots of settings which are worse than another one in terms of both accuracy and latency. So, finding the best setting of control variables is essential in maximizing the system performance.

Simulation. We simulate the executions of Algorithm 1 and Algorithm 2 in the testbed with MEC2 as the server. The resolution and bit rate parameters selected for different frames are shown in Fig. 9, and their corresponding accuracy and latency values are plotted in Fig. 10.

The resolution and bitrate are initially set to the maximum and minimum available values, respectively. For each frame, the sender uses the current parameter setting to stream the video and checks if the receiver uses the same resolution for inference. If so, it means that the current setting does not violate the overall latency constraint and the algorithm continues to increases bitrate for improving accuracy. This can be observed in Fig. 9a that the bitrate follows a steady slope.

In Fig. 10a, both the accuracy and the latency increases for the first few frames. This is because the algorithm is increasing the bitrate while keeping the resolution. At the 9th



(a) YOLOv5s (b) YOLOv5x

Fig. 10: Trace of the searching for the optimal setting, starting from the red dot, each dot represents a frame sequence in Fig. 9

frame, the overall latency is above the latency limit and the algorithm starts to reduce resolution but holds the bitrate. After that, the overall latency in Fig. 10a starts to decrease until getting smaller than the threshold. Finally, the best possible accuracy and latency smaller than 100 ms are reached, as denoted with the cross mark in Fig. 10, which is of the highest accuracy among those having a latency below 100 ms in Fig. 8. This observation verifies the effectiveness of the proposed algorithm, finding the best possible parameter values for each frame. While, in a real-world application, the inference accuracy cannot be obtained in an online manner due to the lack of ground truth labels, one may use probabilistic logic approaches to estimate the inference accuracy as proposed in [23], which is left for future work.

V. Related Work

Streaming's impact on DNN. Some measurement studies have been conducted to understand how the quality of the video itself, e.g., resolution, affects the accuracy of deep object detection [24], [25]. For example, Dodge and Karam [25] figured out that the inference accuracy is susceptible to blur and noise distortions while being resilient to compression artifacts and contrast. In addition, Ran et al. [1] measured how the accuracy changes as a function of latency for videos at a fixed resolution and bitrate. Modern video streaming protocols support adaptation of bitrate for visual quality, our work takes one step further to investigate how the adaptation of video streaming parameters affects jointly the latency and the inference accuracy.

Latency vs accuracy in edge computing. There exist in the literature also several frameworks for deep video analytic at the edge [1], [2]. A key design criterion for such frameworks is to optimize the accuracy of video analytic while minimizing the latency. As most of them apply a data-driven solution, their optimal algorithm is limited to only a single software and hardware setup and is not applicable to other scenarios. Comparing with the existing works, the novelty of our work resides in a generic optimization framework, where no prior knowledge of the application software and hardware is required.

Server-driven video streaming. Traditional video streaming applications focus on visual quality, which adapts bitrate based on network conditions to retain the most visual details for a frame. However, with the emerge of AI applications, where video frames are used for inference instead of visual playback, the application's quality of service (QoS), e.g., inference accuracy, becomes the key metrics in video streaming. So, server-driven video streaming is proposed to optimize streaming for AI application's QoS. For example, DDS (DDN-Driving Streaming) proposes to stream low-quality video at first and let the server determine where to re-send with higher quality [4]. In this paper, we use inference accuracy as a criteria for deciding the best combination of control variables, i.e., streaming bitrate and resolution. Taking advantage of the use of real-time server feedback, our solution works without prior knowledge on the system setup.

VI. Conclusion and Future work

In this paper, we propose an online algorithm that searches for the optimal settings of video streaming parameters, with the aim of maximizing the inference accuracy within a latency constraint. Compared to data-driven methods, the proposed algorithm is adaptive and works without any prior knowledge on the streaming/inference software or the execution hardware. Moreover, our proposal has been proven to converge fast to optimal video streaming settings. In the future, we plan to integrate the proposed algorithm with the transport layer congestion control to maximize application QoS during the change on sending speed.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 825496, and Academy of Finland under grant number 317432 and 318937.

References

[1] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 1421–1429.

[2] T. Tan and G. Cao, "FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile," in *Proc. IEEE INFOCOM*, Jul. 2020, pp. 1947–1956.

[3] M. Ali, A. Anjum, M. U. Yaseen, A. R. Zamani, D. Balouek-Thomert, O. Rana, and M. Parashar, "Edge Enhanced Deep Learning System for Large-Scale Video Stream Analytics," in *Proc. IEEE ICFEC*, May 2018, pp. 1–10.

[4] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-Driven Video Streaming for Deep Learning Inference," in *Proc. ACM SIGCOMM*, Jul. 2020, pp. 557–570.

[5] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live Video Analytics at Scale with Approximation and Delay-Tolerance," in *Proc. USENIX NSDI*, 2017, pp. 377–392.

[6] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (WebRTC)," in *Proc. MMSys*, May 2016, pp. 1–12.

[7] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, A. Chaurasia, L. Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Durgesh, F. Ingham, Frederik, Guilhen, A. Colmagro, H. Ye, Jacobsolawetz, J. Poznanski, J. Fang, J. Kim, K. Doan, and L. Y. , "ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration," Jan. 2021.

[8] D. Xu, A. Zhou, X. Zhang, G. Wang, X. Liu, C. An, Y. Shi, L. Liu, and H. Ma, "Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consumption," in *Proc. ACM SIGCOMM*, 2020, pp. 479–494.

[9] C. Cola and H. Valean, "On multi-user web conference using WebRTC," in *Proc. IEEE ICSTCC*, Oct. 2014, pp. 430–433.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *Proc. IEEE ICCV*, 2015, pp. 1026–1034.

[11] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving," in *Proc. IEEE CVPR*, 2017, pp. 129–137.

[12] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial As Deep: Spatial CNN for Traffic Scene Understanding," in *Proc. AAAI Conf. Artificial Intelligence*, 2018, pp. 7276–7283.

[13] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767 [cs]*, Apr. 2018, arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767

[14] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," in *Proc. IEEE/CVF Conf. Comp. Vision Pattern Recognition*, 2020, pp. 2446–2454.

[15] Y. O. Sharrab and N. J. Sarhan, "Detailed Comparative Analysis of VP8 and H.264," in *Proc. IEEE International Symposium on Multimedia*, Dec. 2012, pp. 133–140.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE TPAMI*, vol. 39, no. 6, pp. 1137–1149, 2017.

[17] E. Kapassa, M. Touloupou, A. Mavrogiorgou, and D. Kyriazis, "5G & SLAs: Automated proposition and management of agreements towards QoS enforcement," in *Proc. IEEE ICIN*, Feb. 2018, pp. 1–5.

[18] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, Oct. 2016.

[19] G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar, "Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving," in *2017 IEEE Region 10 Symposium (TENSYMP)*, Jul. 2017, pp. 1–6.

[20] A. Canziani, A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," *arXiv:1605.07678 [cs]*, 2016.

[21] M. F. Tuysuz and M. E. Aydin, "QoE-Based Mobility-Aware Collaborative Video Streaming on the Edge of 5G," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7115–7125, Nov. 2020.

[22] T. Na, M. Lee, B. A. Mudassar, P. Saha, J. H. Ko, and S. Mukhopadhyay, "Mixture of Pre-processing Experts Model for Noise Robust Deep Learning on Resource Constrained Platforms," in *Proc. IEEE IJCNN*, Jul. 2019, pp. 1–7.

[23] E. A. Platanios, H. Poon, T. M. Mitchell, and E. Horvitz, "Estimating Accuracy from Unlabeled Data: A Probabilistic Logic Approach," May 2017. [Online]. Available: http://arxiv.org/abs/1705.07086

[24] M. Aqqa, P. Mantini, and S. K. Shah, "Understanding How Video Quality Affects Object Detection Algorithms," in *Proc. VISIGRAPP*, 2019, pp. 96–104.

[25] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *Proc. IEEE QoMEX*, 2016, pp. 1–6.