VPPlus: Exploring the Potentials of Video Processing for Live Video Analytics at the Edge

Junpeng Guo
Department of Computer Science
Purdue University
West Lafayette, IN, USA
guo567@purdue.edu

Shengqing Xia
Department of Computer Science
Purdue University
West Lafayette, IN, USA
xia170@purdue.edu

Chunyi Peng
Department of Computer Science
Purdue University
West Lafayette, IN, USA
chunyi@purdue.edu

***Abstract*—Edge-assisted video analytics is gaining momentum. In this work, we tackle an important problem to compress video content live streamed from the device to the edge without scarifying accuracy and timeliness of its video analytics. We find that on-device processing can be tuned over a larger configuration space for more video compression, which was largely overlooked. Inspired by our pilot study, we design `VPPlus` to fulfill the potentials to compress the video as much as we can, while preserving analytical accuracy. `VPPlus` incorporates two core modules – offline profiling and online adaptation – to generate proper feedback automatically and quickly to tune on-device processing. We validate the effectiveness and efficiency of `VPPlus`using five object detection tasks over two popular datasets; `VPPlus` outperforms the state-of-art approaches in almost all the cases.**

***Index Terms*—Video analytics, edge computing, on-device processing, video compression**

I. Introduction

Video analytics is becoming part of the norm. With recent rapid advances in deep learning and edge computing, video content is transmitted and even live-streamed from in-field cameras to edge machines to perform computer vision tasks such as detecting/tracking target objects (e.g., pedestrians, cars, traffic lights, and stop signs) and identifying/counting events of interest (e.g., traffic violations, breaks-in, package deliveries, and suspicious activities in restricted zones). Such analytics on video enriches many new and essential services like security monitoring, autonomous driving, incident detection, subject counting, public safety, and surveillance services across business vectors [1]. Unsurprisingly, global video analytics market is roaring up, with a projection of $21.8B by 2027, quadrupling its size of $5.2B in 2020 [2].

High demands for video analytics pose new challenges in capturing, processing, transmitting, and analyzing quality video. The status quo video analytics runs over deep neural networks (DNNs) and requires heavy computing, which hardly fits on resource-constrained devices. To this end, edge-assisted video analytics becomes one dominant solution paradigm by offloading compute-intensive analytical tasks to the edge [1]. Fig. 1 shows its typical workflow in four or five steps. The device camera first captures raw video content (❶) and then runs on-device processing to convert these raw frames into a bitstream which will be sent to the edge (❷). At the edge side, the received bitstream is decoded and reversely constructed into video frames (❸), which will be eventually fed into the DNN-based module to perform the designated analytical tasks (❹). To make live video analytics possible and more efficient, step ❺ is often added to send real-time feedback (say, intermediate results or other hints generated out of the previous frames at ❹) to guide accuracy-preserving processing (❷) to use fewer bytes for the subsequent frames.

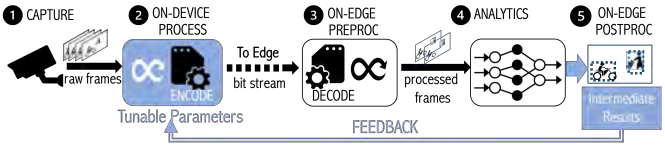


Fig. 1: A typical architecture and workflow of edge-assisted video analytics. Components in blue are the core of our `VPPlus` design.

State-of-the-Art. Recent studies are active in exploring different types of feedback to enhance edge-assisted video analytics (detailed in §II). Generally, they are divided into two categories using implicit feedback and explicit feedback. The implicit feedback is the intermediate results of DNNs like the regions of interest (RoIs) of target objects or events. The implicit feedback is the intermediate results of DNNs like target objects or events' regions of interest (RoIs). It is used to assist the device in transmitting the selected RoIs (subframes) at high quality and non-RoI subframes at low quality [3]–[9]. The device tunes the video size (of the bitstream) by adjusting popular parameters like frame per second (FPS), resolution (R), quantization parameter (QP, a key video codec parameter), or hybrid (e.g., FPS, R, and QP together). In the second category, best K configurations are directly returned to control on-device processing while meeting the accuracy requirement (say, F1 score higher than a given threshold) [10]–[12]. Such best configurations cover a small set of tunable parameters like FPS, R, QP, and DNN models; they are obtained by periodically profiling a short video clip in the near term. A special case is to ask the device to transmit only the selected frames and discard uninteresting frames, e.g., [13]–[16], where the feedback is used to control FPS explicitly.

Despite their effectiveness, the existing studies fail to exploit

978-1-6654-6824-4/22/$31.00 ©2022 IEEE

the full power of on-device processing. As a matter of fact, video quality, as well as its volume size, can be tuned by many more parameters except these popular ones like FPS, R, and QP. For example, each video frame size changes when adjusting its color, brightness, sharpness, edge contrast, blurring (spatial smoothing), and so on. In addition to intra-frame processing, inter-frame processing on motion extraction and cross-frame prediction also impacts the video size (detailed in §III). A larger number of parameters are available to compress video more while not hurting the accuracy, but their potentials are largely unexplored. The second limitation is that the existing feedback is often generated with humans in the loop, which impedes full autonomy for prompt feedback in real-time. Most studies use F1 scores or something alike to assess their inference accuracy. However, these accuracy results require the ground truth labels, which have to take huge manual efforts for training. This limits the power of this feedback for truly live video analytics.

Our `VPPlus` work. We devise `VPPlus` to address the above two limitations and attempt to make full use of video processing for more efficient video analytics. We tackle two technical problems. First, *what parameters are useful, and how should they be used for on-device processing?* Second, *how should feedback be automatically constructed from intermediate results to configure parameters for on-device processing without humans in the loop?*

To answer them, we have first conducted a pilot study to assess the accuracy and efficiency of a broader range of on-device processing parameters (here, 19) in typical use scenarios (§III). We have gained new design insights. For example, some parameters (like brightness and contrast) are able to save big room for efficiency, but their gains are quite sensitive to video content. In other words, there are no universal rules to tune these parameters, and online adaptation is thus needed to pursue a higher level of efficiency. Inspired by these findings, we thus design `VPPlus`, which combines the online outputs of DNNs and the offline profiles to generate configuration parameters for on-device processing as the feedback from the edge to the device. Offline profiling is to learn common patterns which exhibit similar characteristics over diverse video sources. Online algorithms are to fine-tune the parameters based on the real-time impacts on near-term video frames (including the target objects and background). To free humans out of the loop, our online algorithms leverage the intermediate results (actually, the confidence levels) of object candidates as an implicit accuracy measure, instead of the explicit accuracy results which require manual efforts for the ground-truth labelings. They work together to complete a feedback loop with configuration parameters for the subsequent on-device processing in real-time, as illustrated in Fig. 1 and §IV.

Contributions. We have made three main contributions.

• We have conducted a pilot study to reveal the potentials of larger configuration space for efficient video analytics (§III).

• We have designed `VPPlus` to support a broader range of configuration parameters to tune on-device processing for fewer bytes and comparable analytical accuracy (§IV).

• We have implemented and evaluated `VPPlus` with five object detection tasks using two popular datasets (§V). `VPPlus` is effective and efficient in compressing more while preserving accuracy. It outperforms the start-of-art approaches.

Related Studies	Feedback Type	Common Para.			# of Para.
		FPS	R	QP	
[12], [22]–[24]	No feedback	√			1
[25]			√		1
[10], [11], [26]	Explicit	√	√		2
[13]–[16]		√			1
[4], [5]	Implicit		√		1
[6]–[8]				√	1
[3]			√	√	2
[9]		√		√	2
`VPPlus`	**Explicit**	√	√	√	**19**

TABLE I: Comparison of related studies and our `VPPlus` work.

II. Related Work and Their Limitations

DNN-based analytics. Analytics (❹) is the core module of edge-assisted video analytics. Its main task is to automatically recognize temporal and spatial events in video frames for different purposes like object classification, detection, tracking, semantics segmentation, human action recognition, and so on. This is a very active area with many well-established algorithms and models (see a survey [17]). All mainstream solutions are based on DNNs, precisely convolutional neural networks (CNNs). The popular models include YOLO [18] (latest YOLOv5 [19]), Faster R-CNN [20], transformers [21], etc. All these models must be trained offline first. The inference accuracy often drops if training samples cannot cover the input video frames (images) well. In this work, we apply the existing analytics solutions and work on an orthogonal problem that explores the power of feedback to improve on-device processing for higher efficiency.

Feedback-guided on-device processing. Efficiency is mainly realized by on-device processing (❷) with or without feedback from the edge (❺). Generally, there are three cases: (1) no feedback, (2) explicit feedback to configure on-device processing, and (3) implicit feedback to assist on-device processing.

Without any feedback, the first approach runs basic processing locally to filter out some frames, for example, by running a light DNN (e.g., Tiny-YOLOv3) [22], [24], ensuring significant changes in continuous frames [23], [25], or applying heuristic rules [12]. Their efficiency gain is quite limited because the optimization is coarse-grained and slow to react.

The second approach is to give configuration parameters that directly control on-device processing [10], [11], [26]. Their main idea is to crop a long video into several short clips and periodically profile the first few seconds of each video clip to find the best K configurations that satisfy the accuracy requirement. A special case is to tune FPS by explicitly telling the device to transmit only the selected frames [13]–[16]. Due to inherent delay (more than several seconds), explicit

feedback can not return in real-time. Moreover, few parameters (FPS and R) are considered.

The most popular approach is to use intermediate results of DNNs as implicit feedback to on-device processing. The common intermediate results are RoIs, which mark the bounding boxes of the target objects or events. Given RoI information, on-device processing divides a full frame into multiple subframes and transmits only the selected subframes (RoIs) at high quality through adaptive resolution [4], [5], dynamic QP [6]–[8], or hybrid (adjusting FPS and QP together [9], adjusting R and QP [3]). It also considers only a small set of popular parameters, FPS, R, and QP. Moreover, RoIs offer spatial information of the frames but do not carry context information. However, we later will show that a lot of parameters are context-sensitive, which cannot be tuned without proper feedback.

Limitations. These existing studies (Table I) have two major limitations. First, they have explored only a small set of tunable parameters and leave most unexplored. However, we will later show that more than 8 parameters can be used to reduce the video volume size without affecting its analytical accuracy (§III). Second, the widely used feedback cannot support real-time tuning of a wide range of parameters. Specifically, most explicit feedback is generated over the accuracy results (e.g., F1 scores, recall rates), which require ground-truth labeling and cannot be done in real-time without humans in the loop. Implicit feedback generated from the region-based information fails to provide enough context information to tune those context-sensitive parameters (e.g., brightness and sharpness). Our `VPPlus` work attempts to address both limitations. We will elaborate on how `VPPlus` selects candidate parameters out of large configuration space (here, 19) and tunes their values to achieve a better trade-off between accuracy and compression with no need for human involvement (§IV).

III. Pilot Study

In this section, we present a pilot study to assess the impacts of a larger amount of video processing parameters on the cost-effectiveness (size-accuracy) of video analytics. We find that there exists substantial design room which is unexplored.

Experimental settings. We choose *object detection*, one of the most popular analytical tasks, to assess the impacts of video quality on its detection accuracy. We consider three target objects: `airplane`, `car`, and `cat`, which represent three detection difficulty levels from easy to hard. The difficulty level is determined by the characteristics (say, shape, motion, and background) of target objects: (1) easy if the object in a simple background (example: airplane), (2) medium if the object with regular motion and in a complex background (example: car), and (3) hard if the object with irregular motion and in a complex background (example: cat). We use a popular dataset (ImageNet VID 2017 [27]) and randomly select over 50000 frames out of 160 videos for our study. The DNN model uses `YOLOv5s` from the PyTorch model zoo [28], which has been trained for the selected object detection tasks.

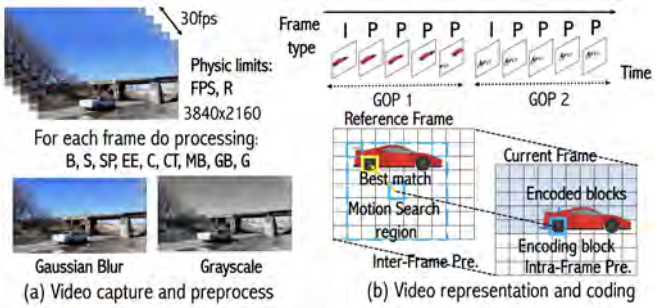


Fig. 2: A large video processing space.

	Parameter	Parameter Values	Explanation
`FPS`	Frame rate	$\{1, 2, 3, 6, 10, 30\}$	30:max FPS
`R`	Resolution	$\{0.25, 0.5, 0.75, 1.0\}$	1= original/ max resolution
`H`	Hue	$\{red, green, blue\}$	
`S`	Saturation	$\{0, 0.25, 0.5, 0.75, 1, 1.5, 2, 3\}$	
`B`	Brightness	$\{0.25, 0.5, 0.75, 1, 1.25, 1.5\}$	0=black. `B` ↑, luminance↑
`G`	Grayscale	$\{true, false\}$	
`SP`	Sharpness	$\{0, 0.5, 1, 2, 5\}$	1=original. `SP` ↑, sharpness ↑
`EE`	Edge Enhance	$\{true, false\}$	
`C`	Contour	$\{true, false\}$	
`CT`	Contrast	$\{0.25, 0.5, 0.75, 1, 1.25, 2\}$	`CT`↑, contrast ↑
`MB`	Median blur	$\{3, 5\}$	`MB`↑, blur ↑
`GB`	Gaussian blur	$\{0.25, 0.5, 0.75, 1, 2\}$	`GB`↑, blur ↑
`GoP`	Group of pictures	$\{50, 100, 200, 350, 400\}$	`GoP`↑, number of I-frames ↓
`NRF`	# of refer. frames	$\{2, 4, 6, 8, 10\}$	
`MSR`	Motion search range	$\{8, 16, 24, 32\}$	
`IS`	Intra smoothing	$\{true, false\}$	
`CIP`	Constrained predict.	$\{true, false\}$	
`CRF`	Constant rate	$\{15, 19, 23, 27, 31, 35\}$	`CRF`↑, size↓
`QP`	Quantization	$\{15, 19, 23, 27, 31\}$	`QP`↑, size↓

TABLE II: Parameter settings in the pilot study.

We consider all the common parameters available to impact video processing, as illustrated in Fig. 2. Table II lists their values tested in our study. The raw video is captured at the maximum FPS (e.g., 30 fps) and maximum resolution (here, 4K, 3840x2160 in Fig. 2). Note that the maximum resolution of the raw videos from the dataset is much lower, mostly $<$ 1K. Possible video processing consists of a series of image transformations/filters (that impact raw pixels at each frame) and video encoding, which converts pixels into a bitstream. We use H.264 as the default video codec and use `FFmpeg` [29] to encode the processed video frames. In total, we consider 19 parameters, which all are configurable through open APIs and do not need to revise the implementation codes.

There are 12 parameters which are not used in video codec but directly transform pixels in each video frame.

- Frame rate (`FPS`) and resolution (`R`) are used to downsample video frames temporally and spatially.
- Hue (`H`), saturation (`S`), and brightness (`B`) are related to color attributes perceived by human vision and impact the tone, intensity, and luminance of a color.
- Grayscale (`G`) is a special image processing which reserves the intensity of light only.
- Sharpness (`SP`), edge enhance (`EE`), contour (`C`), and contrast (`CT`) affects the distinguishability of an object. Especially, `SP` and `EE` are used to enhance the edge contrast, `C` highlights

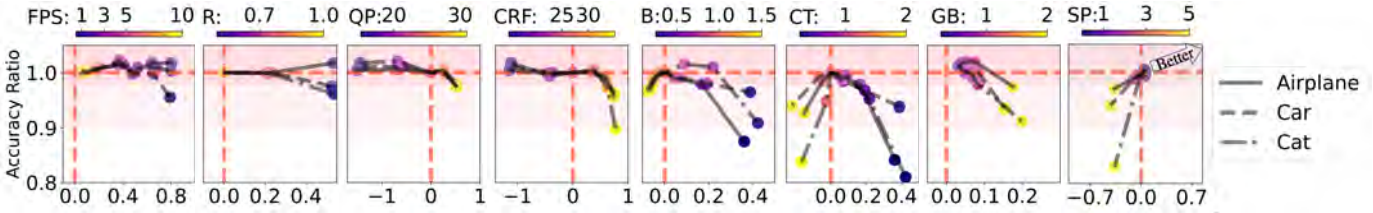


Fig. 3: Normalized accuracy ratios (y-axis) and compression gains (x-axis) of video processing tuned by eight parameters each (from left to right): FPS, R, QP, CRF, B, CT, GB, S. In each detection task (airplane, car, or cat), each point is the average result of all the test video sources when only one parameter setting changes (the value is colored).

the skeleton, and CT is difference in colors.

• Median blur (MB) and Gaussian blur (GB) are used to remove the noise. MB uses the median of all the pixels in the kernel area to replace the central one, and GB uses a Gaussian function to blur the image.

Seven parameters used by video codec are considered. Note that frames are encoded as different types: I-frame and P-frame, as shown in Fig. 2(b). An I-frame is a complete image, while a P-frame (a predicted picture) holds only the changes in the current frame from the previous frame. The video codec often allowds several parameters to tune its inter-frame prediction (motion search) and intra-frame prediction. Finally, it uses transform encoding, which converts pixel values per block into a set of transformation coefficients that are further quantized to compress more.

• Group of Picture (GoP) refers to the number of all the frames between two adjacent I-frames.

• Number of reference frames (NRF) and motion search range (MSR) are the number of reference frames and the region size of nearby pixel exploration for motion search. A larger value means more performed for motion search.

• Intra smoothing (IS) and constrained intra prediction (CIP) are two flag parameters to refine intra-frame prediction. IS is to perform bilinear interpolation to prevent the banding artifacts in the edge. CIP is set to block the error propagation from false inter-frame prediction to intra-frame prediction.

• Constant rate factor (CRF) is a knotting parameter to control compression degrees among different frames.

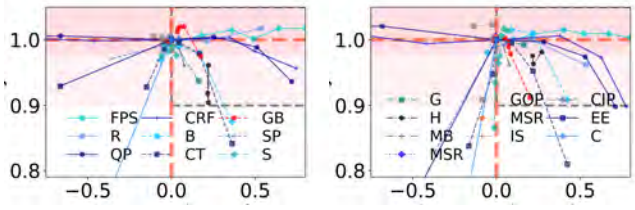
• Quantization parameter (QP) is the last parameter to regulate how much spatial details are reserved. When QP is very small, almost all the details are retained.

We measure cost-effectiveness using two metrics: accuracy and video volume compression (cost). To reveal the trend of different parameters in different detection tasks, we use the normalized ratios instead of their absolute values,

$$\gamma_A = \frac{\text{A[PROCESSED]}}{\text{A[BASELINE]}}, \quad (1)$$

$$\delta_V = \frac{\text{V[BASELINE]} - \text{V[PROCESSED]}}{\text{V[BASELINE]}}. \quad (2)$$

Here, A[BASELINE] and V[BASELINE] are the accuracy and video volume size of the baseline strategy, which uses the original (default) settings and encodes the test video snippets via H.264. The PROCESSED ones are obtained with extra processing tuned by each test parameter. γ_A and δ_V assess the accuracy ratio and compression gain (loss if negative).



(a) Airplane (easy) (b) Cat (hard)

Fig. 4: Accuracy-compression assessment of all the parameters in two tasks: airplane detection (easy) and cat detection (hard).

		FPS	R	QP	CRF	B	CT	GB	SP
Airplane (easy)	**A**	80%	53%	52%	48%	17%	17%	8%	5%
	B	80%	53%	72%	74%	28%	20%	18%	5%
Car (medium)	**A**	65%	23%	52%	35%	19%	7%	13%	6%
	B	65%	52%	71%	75%	38%	38%	13%	6%
Cat (hard)	**A**	79%	21%	29%	38%	22%	9%	7%	6%
	B	79%	64%	53%	77%	42%	22%	20%	6%

TABLE III: The optimal compression gains observed while meeting $\gamma_A \geq 98\%$ (A) or $\gamma_A \geq 90\%$ (B).

Results. We test all the 19 parameters individually and plot the results of eight parameters in Fig. 3. Due to space limit, we choose these eight parameters because they offer greater video compression potentials without compromising accuracy. The rest parameters either compress much less (no more than 5%) or hurt accuracy too much (e.g., C and EE even result in a more than 40% drop in accuracy). To better visualize their impacts on accuracy, we define two regions, A and B, with the dividing lines $\gamma_A \geq 0.98$ and $\gamma_A \geq 0.9$. Accuracy is not affected in region A and declines slightly but is tolerable in region B. To compare their compression potentials, we plot the tunable space of all the parameters in Fig. 4. We use two tasks (airplane and cat) and skip the car detection task due to space limit. A parameter holds the potential to compress more without hurting accuracy too much, as long as there exists at least one point in Region A. We further define the maximal compression potential per parameter as its optimal gain observed in region A (or B). Table III shows the optimal compression gains observed in our study. Parameters are sorted in the descending order of their optimal compression gains in region A in the easy (airplane) detection task. We present the top-8 results due to space limit. We have four main observations.

First, there is no surprise that these well-explored parameters (FPS, R, and QP) yield huge compression potentials. They

are top-3 parameters which compress up to 80%, 53%, and 52% without hurting accuracy ($\gamma_A \geq 0.98$, region A) in the easy detection task (airplane). FPS is consistently effective in all three detection tasks, but R and QP perform slightly worse when the tasks are harder. For instance, R's maximal compression gain reduces from 53% (airplane) to 23% (car) and 21% (cat). It is easy to understand as the detection is harder and more visual details are needed. As a result, there is less compression room to adjust the resolution while retaining the same/similar accuracy.

Second, there exist other parameters which contribute to comparable compression gains. CRF (constant rate) can save more than 35% in all three tasks, which even performs better than R and QP in car/cat detection. B and CT are other two parameters which are worth exploring. They can reduce volume by 7–22% with $\gamma_A \geq 0.98$. If more accuracy drops can be tolerable (region B), their compression gains increase to 20-42%. Note that region B is a relaxed boundary allowing the inherent variance of the detection difficulty among different video sources or video frames. It opens enough room for the online adapatation in the VPPlus design (§IV-B).

Third, we observe win-win patterns in some cases where the accuracy grows, and video volume size reduces. By checking the northeastern zones in Fig. 3, we see that FPS, R, B, and GB can achieve both $\gamma_A > 1$ and $\delta_V > 0$. We would like to point out that although GB (Gaussian Blur) does not yield superior compression gains, it can improve accuracy while reducing the video volume size.

Last, there are no universal rules to tune parameters for cost-effective video processing (compression). Most parameters have distinct accuracy-compression curves across three tasks, actually even across different frames of the same video source in the same task. This indicates that the rule can not be simply learned offline, and online adaptation is required to pursue larger compression room without hurting accuracy.

IV. VPPlus: Exploit More to Save More

We design VPPlus to exploit the potentials exposed in the pilot study. Fig. 5 shows VPPlus's architecture and workflow. At runtime (online), VPPlus changes two modules in edge-assisted video analytics: on-device processing (❷) and edge-assisted feedback (❺). Several algorithms run online to generate real-time feedback based on the near-term inference results of DNN, with the help of configuration rules gained offline. Offline profiling is performed in advance to learn the candidate parameter sets by characterizing their cost-effectiveness, which will be used to assist the online components to in adapting to specific video content. The feedback is returned to the device to guide on-device video processing to compress more without impacting analytical accuracy.

A. Offline Profiling

Offline profiling is to construct a number of useful configuration rules that are commonly effective in reducing the video volume while retaining accuracy. Moreover, the generated profiles should contain a set of configuration choices that offer

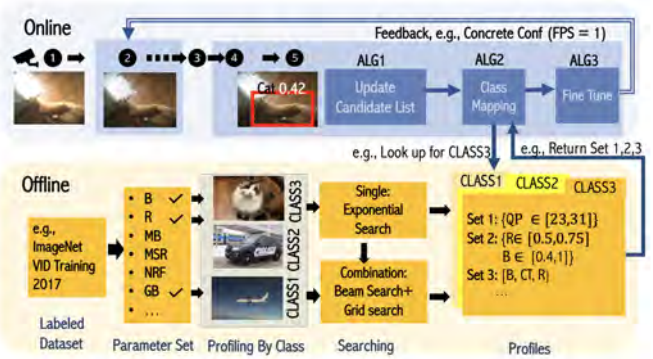
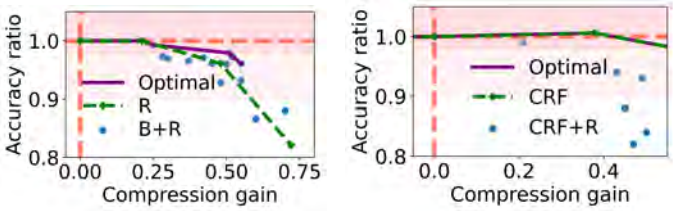


Fig. 5: VPPlus's architecture and workflow.



(a) B + R (b) CRF + R

Fig. 6: Illustration of compound effects. The Pareto optimal points are selected as profilers.

enough room for fine-grained adaptation, which is task-aware and video-dependent, performed at runtime (online).

We only assess 19 parameters individually in §III. Intuitively, multiple parameters can be used together to compress more if they work in concert. This depends on the interleaving effects among these parameters, where they may complement or intervene with each other. Next, we will characterize and learn the compound effects in practice and determine how to select the suitable combinations.

It is not new. [10] has made early attempts and observed that FPS could be tuned independently of resolution (R). This can be easily extended to all intra-frame parameters since FPS does not change the quality of each frame. However, most intra-frame parameters are not independent and jointly determine what information is retained in the color space. In this work, we first tune FPS separately and then consider adding other parameters for more compression.

We use Fig. 6 to illustrate the compound effects. Each blue point represents the result of one combinational setting (left: B + R and right: CRF + R). In the B + R example, we see that using both performs better than using it alone. However, it does not hold in the CRF + R example. The combination of CRF and R can compress more, but the accuracy drops too much. The Pareto optimal points are obtained through searching across all the possible combinations in the training set. They are set as offline configuration profiles.

To accelerate the search for offline profiles, we propose a novel strategy by combining the beam search and grid search as shown in Fig. 7. Intuitively, the idea is similar to the widely used greedy search algorithm: Find the optimal parameter to achieve the highest compression degree without

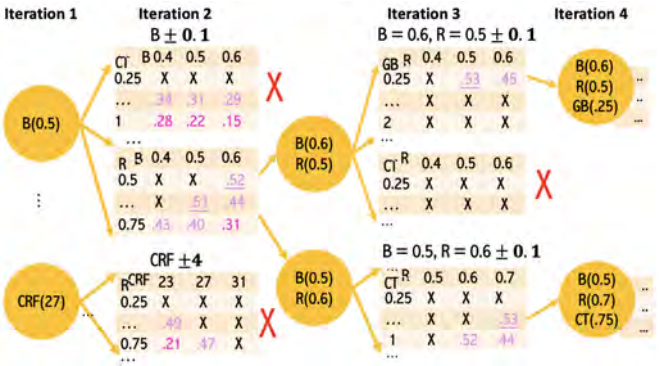


Fig. 7: Illustration of offline profiling (Example: cat detection). Light pink indicates that the configuration results in the result in region B while dark pink for region A.

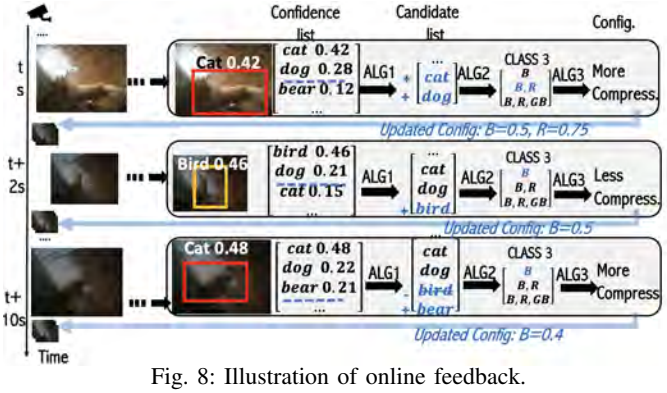


Fig. 8: Illustration of online feedback.

hurting accuracy, and then add one more parameter to locate the best configuration that yields more data reduction. Iterate this process until covering all the parameters or the accuracy drop is no longer tolerable.

Beam search. Beam search algorithm is used to greedily select the top-K optimal candidates for each iteration (represented by circles in Fig. 7). The reason for choosing multiple candidates is that the best candidate might be suitable for the current step, but when we combine it with the other parameters, it may be a suboptimal choice. Instead, the set of top-K best parameters tends to remain stable over time. In practice, we select K = 2. Take Fig. 7 as an example. At the first iteration, we typically select the best 2 parameters, `CRF` (27) and `B` (0.5), as the base and search for other combinations. Note that changing `CRF` is essentially changing `QP`, but it allows different `QP` values to be set within a certain range. Thus we do not consider `QP` here even it can save more than `B`. Only the selected parameters will be considered at the next iteration.

Gird search. Grid search algorithm is used to construct the combination in each iteration (represented by tables in Fig. 7). Specifically, the selected parameter in the last iteration needs to be considered one level up and down in the current iteration to ensure that the selection does not stick at the suboptimal point. Here, we consider `B`$\pm$0.1, `CRF`$\pm$4. Then we select the one with the largest compression (underlined in the figure) to start the next beam search iteration.

Finally, the sets of parameters are identified and sorted in descending order of their compression gains. We construct profiles for object classes. Each class consists of objects with similar characteristics (like shape, motion, and background determined by use scenarios). Each class uses the above searching procedure to construct its profile, which is stored at the edge to bound the scope of online adaptation.

B. Online Adaptation

The offline profiles cannot be directly used at runtime. They are not enough because the compression power of many parameters varies with video context, as disclosed in the pilot study (§III). The object characteristics change over time in the same video. Even for the same object, different positions and movements result in distinct detection results. Use Fig. 8 as an example. This video captures the movement of a cat indoors. As the cat turns its back to the camera, the detection algorithm may detect it as a dog, a bear, or even a bird. Such dynamics make the best parameters learned in the offline profiling may not perform well. Instead, we have to make the parameter values to catch up with dynamics and diversity in video contexts. We thus develop an online adaptation module to exploit the common rules learned offline to fine-tune configuration for higher compression efficiency.

The fundamental problem is how to capture such dynamics and diversity and their impact on detection accuracy. To get correct detection accuracy results, it often requires the ground truth, which is labeled manually. However, this approach is not suitable for live video analytics. We need to provide feedback automatically without humans in the loop. It is challenging to obtain (estimate) the detection accuracy results without the ground truth.

At a high level, we tackle this problem by maintaining a list of candidate labels, which somehow act as the ground truth label. In object detection, a label is treated as a candidate if the confidence value is larger than a given threshold. This candidate list implicitly infers the possible drop in detection accuracy. We observe that the candidate labels tend to remain stable in a dynamic video context. By tracking the change in the list of candidate labels, we are able to infer whether the current configuration is proper. As a result, this process can be performed automatically, which outputs the configuration feedback to fine-tune local video processing.

Adaptation Pipeline. `VPPlus` periodically refines the configurations using a fixed interval ($T = 2$s by default), illustrated in Fig. 8. The interval is determined by the ability of the detection model. The configurations for models with higher detection capabilities are updated less frequently. Once the result is generated by DNN, we first map all the detected bounding boxes to the corresponding ones in the last processed frame. This mapping is achieved by considering the bounding box locations and the corresponding relationships between the output labels and the candidate lists obtained in the last processed frame. For each bounding box, we first update the candidate list based on the DNN output. According to

Algorithm 1 Candidate List Update

Input: $label_i$, $C_{i_{new}}$, $C_{i_{old}}$ (C denotes as confidence value)
Output: $Candidate$: a list of candidate ground truth labels; $Vote$: a value suggesting how to change configs.
function CANDIDATELIST($C_{i_{new}}, C_{i_{old}},$)
    Initial $Candidate$ as an empty list, $Vote = 0$
    for $label_i$ in $Candidate$ **do**
        $L_i \leftarrow \alpha L_i + \beta W_i * (C_{i_{new}} - C_{thre})$
        if $C_{i_{new}} \leq C_{thre}$ **then**
            if $L_i \geq L_{thre}$ **then**
                $Vote \leftarrow Vote - 2 * L_i$
            else
                delete $label_i$ from $Candidate$
    for $i = 1, 2, \ldots, n$ **do**
        if $C_{i_{new}} \geq C_{thre}$ **then**
            if $label_i$ not in $Candidate$ **then**
                add $label_i$ to $Candidate$
            else
                if $C_{i_{new}} \geq C_{i_{old}}$ **then**
                $Vote \leftarrow Vote + 1 * L_i$
                else
                $Vote \leftarrow Vote + 0.8 * L_i$
    $Vote \leftarrow Vote/len(Candidate)$
    return $Vote$, $Candidate$

Algorithm 2 Class Mapping

Input: $bbox$
Output: $ConfigsSet$: a list of suggesting configuration parameters
function CFGSELECT($C_{i_{new}}, C_{i_{old}}, bbox$)
    $vote$, $Candidate$ = CANDIDATELIST($C_{i_{new}}, C_{i_{old}}$)
    $NLP\_Embedding(Candidate)$ and map to $Class_{j \in \{1,2,3\}}$ with the highest semantic similarities
    if $frameDiff \geq diff_{thre}$ **then**
        update $ConfigsSet$, increase FPS.
    if $bbox \leq bbox_{thre}$ **then**
        update $ConfigsSet$, reduce R tuning range.
    return $ConfigsSet$

Algorithm 3 Fine Tuning

Input: $HistVote$
Output: $Configs$
function CFGTUNE($C_{i_{new}}, C_{i_{old}}, HistVote$)
    $vote$, $Candidate$ = CANDIDATELIST($C_{i_{new}}, C_{i_{old}},$)
    add $vote$ to $HistVote$
    if detect continuously positive $Vote$ **then**
        Aggressively move to the config. with more compression
    else if detect continuously negative $Vote$ **then**
        Aggressively move to the config. with less compression
    else
        adjusting current parameters' values based on $Vote$
    return $Configs$

the candidate list, we can map the video content to the predefined class in offline profiling and get the specific sets of configurations. Last, we fine-tune the configuration to keep up with the dynamic video context. An updated feedback is sent back to guide the on-device processing.

Updating the candidate list. We develop Algorithm 1 to construct and update the candidate list ($Candidate$) by leveraging only intermediate results of DNN. We add an important variable $Vote$ to indicate whether the current configuration is feasible. In turn, the trend on the confidence change of each label impacts $Vote$. $Candidate$ and $Vote$ are used together to determine the configuration. The heuristic idea is to add a big penalty (negative value) to $Vote$ for the case that the label is likely to be ground truth but with a low confidence value, indicating the configuration may mislead the detection result. For each label in $Candidate$, add different rewards (positive value) to $Vote$ based on the changing tendency. If the $Vote$ is positive by traversing the $Candidate$, it indicates a further compression can be made.

Specifically, we introduce a L_i to denote the likelihood of the label being the ground truth to determine how detection may be affected by the configuration. To calculate L_i, we consider the aggregated changes in the confidence value from the past to the present. Particularly, results for frames closer to the current frame should play a larger role, so we add coefficients α, β that recursively reduce the impact of results from further history. Moreover, the more times in history that the confidence value of label i is above the threshold, the higher the probability that this label is ground truth. We use W_i to denote the normalized number of times among the past 5 times detection that the confidence value of $label_i$ (Ci) is over the threshold. Back to Fig. 8, for the frame captured at $t+2$s, the cat is missed detected as a bird, and the confidence value of the cat is dropped to 0.15, below the threshold. We calculate L_i for the cat, showing that the cat is likely to be the ground truth. It will go to line 9 and finally lead to a negative $value$, and less compression suggestion is made. The same judgment goes through for $t + 10$s, and the confidence value of the bird is first below the threshold. However, this time it will go to line 11 and delete bird from $Candidate$ since almost all the results from $t + 2$s to $t + 10$s do not regard the object as a bird. Line 12 to line 20 is used to add the new label into $Candidate$ and based on the trend of confidence value update $Vote$. Specifically, a relatively big reward is given when the confidence value is increased. A small reward is for the label where the confidence value drops but is still above the threshold.

Mapping the video context to the offline profiler. In order to turn the heuristics learned in offline profiling, we use Algorithm 2 to do the mapping to the offline profiler. Two strategies are used accordingly. The first is to leverage the semantic relationship to map the $Candidate$ from Algorithm 1 into three defined categories. The second is to further tune the pre-defined set based on the properties of the video, such as resolution and degree of captured motion. To be specific, We apply average NLP embeddings on the $Candidate$ and map it to the pre-defined class based on the similarity. For instance, in Fig. 8, the embedding of $Candidate$ is always set 1. If it cannot have a determined mapping, the common setting is chosen. We further limit the searching range of R and FPS according to the bounding box size and difference between the frames, respectively. This is because even if we

determine the class of target, the size and moving speed of the target will greatly affect the detection accuracy. If the raw captured video has a high resolution (i.e. 4K) and the output $bbox$ is large, there exists a broader room for R tuning. We can further set R to 0.11. As assessed by [3], [5], changing from 4K to 1K (even 720p) will not affect the accuracy. In contrast, if the $bbox$ is already too small, we will not tune the R at all. Similarly, if the raw captured video contains a lot of motion resulting in a large frame difference, high FPS is considered.

Fine-tuning configuration. Given the dynamics of detection difficulties within the video, we need to tune the configuration accordingly. Algorithm 3 shows how `VPPlus` determines the specific parameter configuration. In total, there are two modes: aggressiveness and conservation. The detailed mode is based on the accumulated DNN behaviors. Specifically, we monitor the $Vote$ from Algorithm 1 for several iterations (here, 3 by default). If $Vote$ is in pure positive(negative), the aggressive mode is triggered. It will not tune the parameter step by step, and will skip two-level configurations. On the contrary, if $Vote$ is hybrid, we can change the value within the currently selected set based on the current $Vote$. Specifically, if $Vote > 0$, the configuration with more compression may be selected; otherwise, the selection will move one step back to the less compressed parameter.

V. Evaluation

We evaluate the effectiveness and efficiency of `VPPlus` using three toy tasks and two real-world detection applications. In all the tasks, `VPPlus` outperforms the state-of-art approaches by saving more bandwidth and processing faster while retaining comparable accuracy.

Datasets and tasks. Our evaluation uses four typical object detection tasks over two popular datasets (ImageNet VID 2017 [27] and OAK [30]), as detailed in Table IV. In addition to three single-object detection tasks (airplane, car, and cat) used in our pilot study, we use the OAK dataset to add two more realistic tasks to detect multiple objects in two complex outdoor environments: campus and downtown. All the used OAK videos are captured by a moving phone used by a walking person. Both detection tasks cover six objects: *car, person, bus, bicycle, trunk, and motorcycle* but with distinct object density (campus: sparse and downtown: dense).

Prototyping and evaluation metrics. We prototype `VPPlus` with a laptop (MacBook Pro 2021) as the device and our lab server as the edge machine. We implement all the steps illustrated in Fig. 1, except video capturing (step ❶). Instead, all the original video frames from the datasets are streamed in chronological order to on-device processing ❷) and then transmitted to the edge for each analytical task ❹). Feedback is sent back at ❷) to guide on-device processing of the subsequent frames. Our objective is to reduce the amount of data, not to address how to transmit it. Therefore, all transmissions are via the laptop-server connection. We choose YOLOv5s [28] as the DNN model, as we do in our pilot study.

Dataset	**ImageNet VID [27]**			**OAK [30]**	
Task	**Airplane**	**Car**	**Cat**	**Campus**	**Downtown**
Difficulty	Easy	Medium	Hard	Harder	Hardest
# of frames	12634	14549	21440	11700	10500
# of videos	34	38	55	10	10

TABLE IV: Datasets and tasks for the evaluation.

We compare `VPPlus` with the BASELINE and two state-of-art solutions: Chameleon [10] and saliency-based design (called it Saliency afterward) [9]. Specifically, Chameleon is the most cited work by giving explicit feedback to configure on-device processing, while [9] is the most recent study which yields a significant compression improvement by using saliency maps as implicit feedback to assist RoI-aware on-device processing. Through offline profiling, we find that downsampling to 1 fps is sufficient to retain comparable accuracy. We thus choose 1 fps as the base setting of `VPPlus`. For a fair comparison, Chameleon and Saliency also use 1 fps (which makes their performance even better).

For a given method, we obtain its accuracy ratio γ_A and compression gain δ_V compared to the BASELINE (the default setting),

$$\gamma_A = \frac{\text{A[Method]}}{\text{A[BASELINE]}}, \delta_V = \frac{\text{V[BASELINE]-V[Method]}}{\text{V[BASELINE]}}.$$

Note that γ_A and δ_V use the same forms as eqn. (1) and (2); A[Method] and V[Method] are the accuracy result and the video size of the given method (`VPPlus`, Chameleon [10], or Saliency [9]). To assess the accuracy-compression tradeoff, we introduce a new metric F_{AV},

$$F_{AV} = (1+\omega^2)\frac{\delta_V \times \gamma_A}{\omega^2 \times \delta_V + \gamma_A.} \tag{3}$$

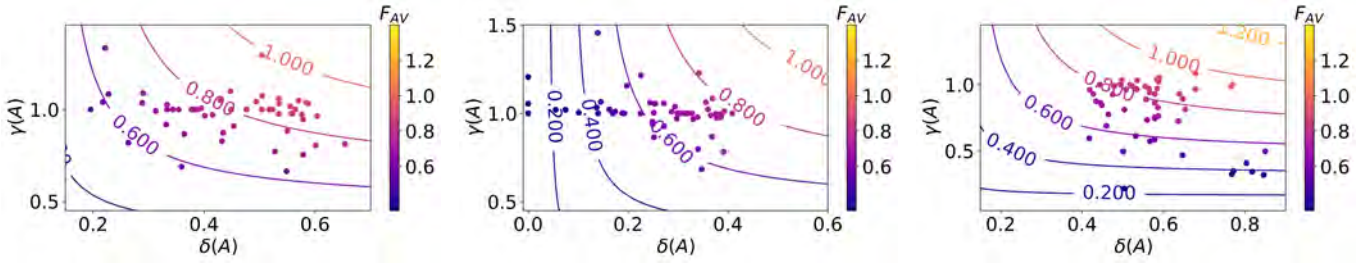
F_{AV} is a weighted score of γ_A and δ_V in a general form, where ω is chosen such that γ_A is considered ω times as important as δ_V [31]. Specifically, γ_A and δ_V contribute equally to F_{AV} when $\omega = 1$. We find that when $\omega = 3$, the score is fully dominated by the accuracy ratio, which underestimates the compression gain. We thus empirically set $\omega = 2$ in our study. A larger F_{AV} score stands for a better compression-accuracy tradeoff.

A. Showcase: Cat Detection

We first use a showcase example to illustrate how `VPPlus` works and demonstrate the benefits of our `VPPlus` solution. In this example, we use a 12-second video clip which captures a cat indoors (Fig. 9). The detection is challenging because the cat turns its back to the camera. Even worse, the cat is black, and its features (object shapes) are hard to distinguish, which lowers The detection results match our expectations. In this example, only frame $t+4$ successfully detects the cat in the BASELINE setting.

In this example, `VPPlus` achieves the mean Average Precision (mAP) of 84.5% while reducing half of the size (from 748KB to 378KB, Table V). We see that both Chameleon and `VPPlus` slightly increase the accuracy, showing that target-specific on-device processing can improve the detection

Fig. 9: Examples of intermediate detection results over the consecutive frames by online algorithms: top (original) and bottom (VPPlus).



(a) VPPlus (b) Chameleon [10] (c) Saliency [9]

Fig. 10: Comparison in the cat detection task: VPPlus outperforms Chameleon and Saliency.

	Volume	mAP	γ_A	δ_V	F_{AV}
BASELINE	748 KB	83.9%	–	–	–
Chameleon [10]	562 KB	85.6%	1.02	0.25	0.63
Saliency [9]	435 KB	78.0%	0.92	0.42	0.74
VPPlus	378 KB	84.5%	1.01	0.50	0.84

TABLE V: Compression and accuracy results in the showcase.

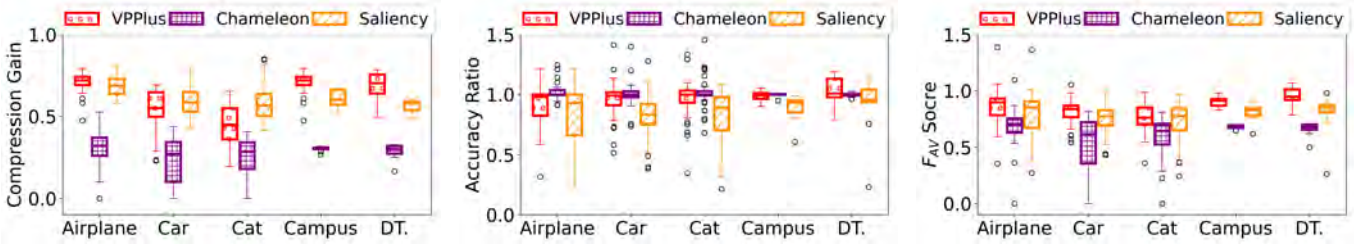
accuracy even when reducing the video quality (volume size). Chameleon maximizes accuracy as it explicitly guides on-device processing by choosing the most appropriate configuration based on F1 score (a metric of precision and recall). However, its compression gain is the smallest (0.25) because only limited parameters (FPS and R) are considered. Salience does compress more (0.42, slightly smaller than 0.50 by VPPlus), even though it only explores FPS and QP only. This is because Salience uses the relatively low-quality frames (here, QP=32) to obtain the saliency maps and utilize them to identify the most appropriate RoIs where the change directly impacts the detection result. Then they send the subframe defined by RoI at a higher quality. However, the saliency map is constructed using the gradient of the output to the input, and its performance depends heavily on the class with the highest confidence value, namely the output label. If the model gives incorrect labels or even does not generate bounding boxes, the defined RoI may be wrong, resulting in a loss of accuracy. In this example, this problem occurs at frame $t+8$. As a result, it hurts detection accuracy more than VPPlus and Chameleon. Overall, VPPlus performs best, which is reflected in its highest F_{AV} score (maximal δ_V and almost best γ_A).

In this example, we would like to emphasize that VPPlus does not require the ground truth but keeps adjusting configuration parameters timely and accurately as if the ground truth were provided as the feedback. To demonstrate our advantage, we calculate the F1 score (calculated based on ground truth) for each processed frame as well in Fig. 11c. Clearly, we see that our configurations are updated consistently with those that use the change in F1 score (ground truth) to directly determine the next configuration. Its effectiveness is validated by constantly increasing F1 scores, which indicates the degree of precision is continuously increased or retained. Regarding compression, the combination of multiple parameters is applied to compress more starting from frame $t+2$; distinct configurations (downsample to a lower resolution and use lower brightness) are set to gradually increase the degree of compression. The compression strategy takes a step back for frame $t+8$ because our model considers current compression is too aggressive. The cat is regarded as a strong candidate, but it has a relatively low confidence value. VPPlus thus downgrades the compression degree for the next frame.

B. In All the Tasks

Cost-Effectiveness. We extend the above case study to a large-scale evaluation and confirm VPPlus outperforms Chameleon and Saliency in all five tasks.

We first show their comparison in the cat detection task (hard) in Fig. 10. Due to space limit, we omit the results in the other three tasks. We have three observations. First, Chameleon performs best in accuracy but fails to compress a lot. Actually, it achieves comparable accuracy with the *BASELINE*. Second, Saliency achieves a large degree of compression at the cost of a larger accuracy drop. There are many instances with $\gamma_A < 0.9$ (or even < 0.5). Third,



(a) Compression gain (b) Accuracy ratio (c) F_{AV} scores

Fig. 11: Comparisons in all the five detection tasks. DT is short for Downtown.

VPPlus achieves a good tradeoff. We notice that there are some instances with a large accuracy drop in both VPPlus and Saliency. This is because both have to implicitly infer the detection accuracy. It is hard to guarantee the correctness of inference without the ground truth. Inevitably it might hurt accuracy in case the detection is really challenging. To fairly compare these methods, we want to compare their performance for most videos, not all the videos. We consider 95% confidence level and see the accuracy ratio of VPPlus is [0.94,1.02], while [0.83,0.95] for Saliency. This implies that VPPlus guarantees most videos do not suffer from a more than 6% accuracy loss. Overall, VPPlus achieves a better accuracy-compression tradeoff in the cat detection task.

Fig. 11 further shows the statistical results in all the five tasks. In terms of F_{AV} scores, VPPlus outperforms the other two in all the tasks. We further take a closer look at where this benefit comes from. On average, the compression gain of VPPlus is 20-50% higher than Chameleon with only a 3-6% accuracy drop. Note that Chameleon uses the ground truth to do periodical profiling, which ensures high accuracy. Compared to Saliency, VPPlus does not compress a lot in three toy tasks but saves a lot in the two real-world detection tasks (campus and downtown). Moreover, Saliency's compression gains are achieved at the cost of accuracy reduction, which results in the worst compression-accuracy tradeoff (namely, the lowest F_{AV} scores). In sum, VPPlus achieves the best compression-accuracy tradeoff, particularly in the two real-world detection applications.

Given the different performances across the tasks, we want to point out the impact of capture setting and video contexts. First, we observe the overall largest compression gain achieved by the OAK dataset. This is because the videos are captured at high resolution (720p), compared with the ImageNet dataset (most 360p). This thus leaves a large room to tune resolution to save more. Second, the compression gain is heavily affected by the video context. In the same-quality raw videos (airplane, car, and cat using ImageNet dataset), more compression can be made when the task is relatively easier because videos have a clean background and the detection is easier. Third, context switches may lead to a loss of accuracy. In the OAK dataset, the contexts are consistent over time for the downtown scenario; however, more dynamic contexts are captured on campus (from the campus building to the street), changing the target objects' distributions. Correspondingly, VPPlus's accuracy preservation (γ_A) is higher in the downtown than the campus one since it needs to use historical data to assist the runtime configuration updates; the delay between the update algorithm and scene switching leads to the drop in detection accuracy. The same situation applies to Saliency. Moreover, we notice that VPPlus's γ_A has a larger variance compared to the rest two methods in the downtown detection task. This is because the mapping procedure is more challenging with dense and tiny object detection.

	YOLOv5	**Saliency [9]**		VPPlus	
	TPF	TPF	(%)	TPF	(%)
Airplane	263 ms	482 ms	83%	274 ms	4.2%
Car	224 ms	449 ms	100%	245 ms	4.6%
Cat	237 ms	470 ms	98%	246 ms	3.8%
Campus	270 ms	510 ms	89%	321 ms	19%
Downtown	277 ms	560 ms	102%	341 ms	23%

TABLE VI: The extra overhead (the average time per frame (TPF)) of VPPlus is small, much smaller than Saliency.

System overhead. We measure system efficiency in terms of the processing time per frame. Table VI shows the average processing time of baseline (YOLOv5), Saliency, and our VPPlus solution. We see that the extra time for VPPlus is small (no more than 5%) in three single-object detection tasks (airplane, car, and cat). Only in the detection in dynamic scenarios (campus and downtown), VPPlus adds about 20% extra time. This is because there are multiple objects in the video frames, which takes more time for object mapping in order to find the matched pairs. We want to point out that our VPPlus solution is much faster than Saliency since the saliency map can only be obtained from backpropagation, which almost doubles the time because YOLOv5 needs to traverse twice (back and forth).

VI. Conclusion and Future Work

In this work, we present our attempt to explore the potentials of video processing to make video analytics more cost-effective. We aim to compress video more without hurting its analytical accuracy. In addition to a small set of well-explored parameters, we find that other parameters are also useful but largely outlooked. We thus design, implement, and evaluate VPPlus. We show that it is promising to exploit this larger space in real-time for truly live video analytics.

There are several remaining or open issues in VPPlus.

Finer granularity. We use the majority principle to determine the configuration for the whole frame when multiple objects are detected. We can further exploit RoIs for fine-grained compression. The technical problem that needs to be solved is how to map to the likely outdated feedback information to identify RoIs since the frame rate we used is relatively low.

Adaptive resolution. Constantly changing resolutions make it difficult to directly use the existing video codec. To use the existing codec, we group the consecutive frames with the same resolution into the video segments. Our future work will focus on determining whether to directly use image transmission or to devise new video technologies for encoding that can be compatible with varying resolutions and frame rates.

New video codec. The community is exploring a brand new way for video codec, which is more friendly to video analytics, like MPEG-7 [32]. The advances are orthogonal to our efforts and can open up new room with new parameters and codec APIs exposed. We leave it as our future work.

More DNN models. We admit that offline profiling should change with new DNN models and training datasets. There are new DNN designs that recently exploit graph neural networks (GNN) (e.g., [33]) and transformer (e.g., [34]). We are extending our work to support these new models. We will extend our work to support these new models in the future.

More analytical tasks. There are many other video analytics tasks such as human action recognition, event counting, and semantical segmenting. We plan to explore the power of a larger configuration space on these tasks in the near future.

ACKNOWLEDGMENTS

We greatly appreciate constructive comments from our anonymous reviewers. This work is supported in part by National Science Foundation (NSF) CAREER Award 1750953 and Adobe. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or Adobe.

REFERENCES

[1] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[2] A. T. Research, "Video analytics market statistics: 2027," April 2021, https://www.alliedmarketresearch.com/video-analytics-market. Last access on Nov 17, 2021.

[3] J. Guo and C. Peng, "Towards live video analytics with on-drone deeper-yet-compatible compression," *arXiv preprint arXiv:2111.06263*, 2021.

[4] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, "Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading," in *MobiCom*, 2021, pp. 201–214.

[5] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, "Flexible high-resolution object detection on edge devices with tunable latency," in *MobiCom*, 2021, pp. 559–572.

[6] X. Wang, A. Chowdhery, and M. Chiang, "Skyeyes: adaptive video streaming from uavs," in *Proceedings of the 3rd Workshop on Hot Topics in Wireless*, 2016, pp. 2–6.

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

[8] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *SIGCOMM*, 2020, pp. 557–570.

[9] N. A. R. N. J. J. Qizheng Zhang, Kuntai Du, "Understanding the Potential of Server-Driven Edge Video Analytics," in *HotMobile*, 2022.

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

[11] K. Wu, Y. Jin, W. Miao, Z. Zeng, Z. Qian, J. Wang, M. Zhou, and T. Cao, "Soudain: Online adaptive profile configuration for real-time video analytics," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021.

[12] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *SenSys*, 2015, pp. 155–168.

[13] H. Meuel, F. Kluger, and J. Ostermann, "Region of interest (roi) coding for aerial surveillance video using avc & hevc," *arXiv preprint arXiv:1801.06442*, 2018.

[14] 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 *NSDI)*, 2017, pp. 377–392.

[15] L. Cheng and J. Wang, "Vitrack: Efficient tracking on the edge for commodity video surveillance systems," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1052–1060.

[16] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2020, pp. 110–124.

[17] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.

[19] G. Jocher *et al.*, "Yolov5," *Code repository https://github. com/ultralytics/yolov5*, 2020.

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[21] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *ECCV*. Springer, 2020, pp. 213–229.

[22] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *SEC*. IEEE, 2018, pp. 159–173.

[23] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *SIGCOMM*, 2020.

[24] G. Ananthanarayanan, Y. Shu, L. Cox, and V. Bahl, "Project rocket platform—designed for easy, customizable live video analytics—is open source. microsoft research blog," 2020.

[25] J. Meng, S. Paul, and Y. C. Hu, "Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices," in *ASPLOS*, 2020, pp. 923–937.

[26] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[27] "Imagenet large scale visual recognition challenge 2017," https://image-net.org/challenges/LSVRC/2017/.

[28] "Yolov5 in pytorch," https://pytorch.org/hub/ultralytics_yolov5/.

[29] "Ffmpeg," https://www.ffmpeg.org/.

[30] J. Wang, X. Wang, Y. Shang-Guan, and A. Gupta, "Wanderlust: Online continual object detection in the real world," in *ICCV*, 2021, pp. 10 829–10 838.

[31] Wikipedia, "F-score," https://en.wikipedia.org/wiki/F-score.

[32] "Mpeg-7," 2022, https://mpeg.chiariglione.org/standards/mpeg-7.

[33] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[34] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.