Color-based Lightweight Utility-aware Load Shedding for Real-Time Video Analytics at the Edge

Harshit Gupta*
Enrique Saurez*
guptaharshit@microsoft.com
esaurez@microsoft.com
Georgia Institute of Technology
USA

Henriette Röger
henriette.roger@ipvs.uni-stuttgart.de
University of Stuttgart
Germany

Sukanya Bhowmik
sukanya.bhowmik@uni-potsdam.de
University of Potsdam
Germany

Umakishore Ramachandran
rama@gatech.edu
Georgia Institute of Technology
USA

Kurt Rothermel
kurt.rothermel@ipvs.uni-stuttgart.de
University of Stuttgart
Germany

ABSTRACT

Real-time video analytics typically require video frames to be processed by a query to identify objects or activities of interest while adhering to an end-to-end frame processing latency constraint. This imposes a continuous and heavy load on backend compute and network infrastructure . Video data has inherent redundancy and does not always contain an object of interest for a given query. We leverage this property of video streams to propose a lightweight Load Shedder that can be deployed on edge servers or on inexpensive edge devices co-located with cameras. The proposed Load Shedder uses pixel-level color-based features to calculate a utility score for each ingress video frame and a minimum utility threshold to select interesting frames to send for query processing. Dropping unnecessary frames enables the video analytics query in the backend to meet the end-to-end latency constraint with fewer compute and network resources. To guarantee a bounded end-to-end latency at runtime, we introduce a control loop that monitors the backend load and dynamically adjusts the utility threshold. Performance evaluations show that the proposed Load Shedder selects a large portion of frames containing each object of interest while meeting the end-to-end frame processing latency constraint. Furthermore, it does not impose a significant latency overhead when running on edge devices with modest compute resources.

CCS CONCEPTS

• **Information systems** → *Data streams*; *Stream management*; • **Computing methodologies** → **Computer vision**.

KEYWORDS

Video Analytics, Load Shedding, latency bound, QoS.

ACM Reference Format:
Harshit Gupta, Enrique Saurez, Henriette Röger, Sukanya Bhowmik, Umakishore Ramachandran, and Kurt Rothermel. 2024. Color-based Lightweight Utility-aware Load Shedding for Real-Time Video Analytics at the Edge. In *The 18th ACM International Conference on Distributed and Event-based Systems (DEBS '24), June 24–28, 2024, Villeurbanne, France.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629104.3666037

1 INTRODUCTION

Real-time video analytics has been gaining rapid popularity due to its utility in applications such as surveillance[1], driving assistance and safety[2], and factory automation[3]. Such applications are typically structured as a pipeline of operators, where each operator executes a piece of the overall application logic, e.g., object detection, classification, activity recognition, etc., and extracts relevant insights from camera frames. We specifically focus on video analytics pipelines with stringent end-to-end latency constraints, such that the extracted insight from video processing could be used to trigger a real-time response, e.g., alert to car driver. The increasing availability of high quality and high frame rate cameras

*Author currently works at Microsoft Corporation.

DEBS '24, June 24–28, 2024, Villeurbanne, France

ACM ISBN 979-8-4007-0443-7/24/06
https://doi.org/10.1145/3629104.3666037

put significant pressure on the backend compute and networking resources. Although the use of edge resources for running geo-distributed video analytics has been proposed to minimize backhaul bandwidth usage [4], the resource capacity of edge sites is typically limited due to space and power constraints [5]. Oftentimes complex operators like object detection pose heavy compute requirements, such as access to a GPU, which imposes limitations on the number of cameras that can be served at a given edge site.

Video streams possess two key characteristics that enable serving more number of cameras with limited resources. Firstly, the appearance of the object-of-interest for a given analytics query is not frequent[6], implying that a large fraction of camera frames do not contain useful information. Secondly, when an object-of-interest exists in a video stream, due to the high frame rate of cameras it usually is present in multiple frames. Dropping a small portion of the frames that an object-of-interest appear in does not affect the overall fidelity of the results. These characteristics motivate the use of load shedding techniques to shed irrelevant frames, to reduce the workload on the application pipeline. Previous work in load shedding has focused on using linear selectivities [7] and work with structured queries and data [8, 9]. Such techniques haven't been explored for content-based shedding of unstructured data such as video. Previous work in early-discard of video frames either require expensive hardware for feature extraction [10, 11] or do not tune the filtering parameters according to the processing load on the backend video analytics pipeline [12].

In this work, we present a lightweight load shedding technique that uses a per-query content-based utility function to determine if a frame should be shed. The utility of a frame is calculated as a function of its color distribution. Each query undergoes a learning phase during which the utility function is built. The Load Shedder receives all ingress frames and it drops those whose utility is below the baseline utility threshold. Due to inherent variations in video streams' contents, the utility threshold needs to be dynamically tuned so that the load on backend analytics pipeline is within manageable levels, and the end-to-end processing latency constraint of the query is continuously met. The Load Shedder includes a feedback control-loop that dynamically updates the utility threshold based on the current load on the later stages of the video processing pipeline. This feedback from the later stages ensures that the overall query processing pipeline functions correctly despite differences in the content of the video stream compared to the training set. We incorporate the proposed load shedding technique on a video analytics platform and perform extensive evaluations with real-world analytics queries and video datasets. Our contributions can be summarized as follows:

- A workflow for building the per-frame utility function, given a query and a labeled training data set. The lightweight utility function processes a high rate of ingress frames without imposing significant latency overhead.
- A control loop that dynamically tunes the utility threshold based on the current query operator load, thus keeping the end-to-end latency under a query-specific bound.
- Performance evaluation of the proposed load shedding approach to demonstrate its efficacy.

2 BACKGROUND AND PROBLEM STATEMENT

We, first, describe the context of our proposed approach, followed by a formal definition of the problem statement.

2.1 Context

This section sets the context for the proposed contributions, in terms of the target application scenarios and representative deployment scenarios for the proposed system.

2.1.1 Target Application Scenarios. We target real-time video analytics queries for which the target objects can be described using a specific set of colors. Such queries are common in the domains of surveillance (e.g, tracking red cars in response to an AMBER alert [13]), traffic control (e.g., detecting if an emergency vehicle is stuck in traffic [14]), search and rescue (e.g., locating humans in open water using drones [15]), etc. Such queries typically process multiple frames containing a given target object (e.g., a suspicious red car in first example) to extract insights about the object (such as its direction of motion, or which street it went to from an intersection). The query could be designed to process frames containing target objects of a single color (e.g., red suspicious vehicle), frames containing at least one object from the target colors (e.g., containing either a white ambulance or a red fire truck) or frames containing objects of all target colors (e.g., frames containing both a red car and another white car).

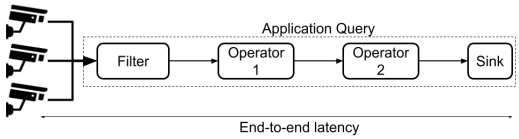


Figure 1: High-level query model: proposed filter module filters the camera streams processed by the query's operators.

A high-level query model is composed of three components: a filter, a sequence of one or more video processing operators (e.g., Deep Neural Network based object detection), and a sink (Fig. 1). The filter discards frames with no useful information - for instance, dropping frames without contiguous groups of pixels (blobs) of a certain color larger

than a certain size. The video processing operators form the core of the query logic. Finally, the sink analyzes the output labels of the video processing operators and sends information downstream, either to perform an action or to execute further downstream video processing queries. Depending on the dynamic content of video streams, the fraction of frames processed by the video processing operators (e.g., heavyweight DNNs) and the sink varies with time. This variation in processing load causes significant variation in end-to-end latency of the system as well. To ensure real-time response, the queries have a constraint on end-to-end latency of processing a frame containing a target object. The end-to-end processing latency of a frame is the total time taken between the generation of the frame by the camera to the time when it is executed completely by the application query (including communication delay).

2.1.2 Representative Deployment Scenarios. We consider a connected camera deployment which could (but not necessarily) be assisted by an edge compute node available in proximity. We assume cameras to contain limited compute capability for running background subtraction and feature extraction (to be discussed in Section 4.2). Cameras send the foreground of frames along with the associated features downstream. The possible scenarios of the deployment of downstream components, i.e., the Load Shedder and Application Query, are shown in Fig. 2. In each such scenario, either the compute resource on the edge or the network bandwidth between edge and cloud or between camera and cloud is the bottleneck resource, whose over-utilization results in excessive queuing of video frames, eventually resulting in violation of the end-to-end processing latency constraint. Hence, the Load Shedder's effective operation is crucial to make sure that the bottleneck resource is not overloaded. The objective of the Load Shedder is to maximize the fraction of frames containing each target object that are sent downstream to the application query after shedding. It optimizes this objective while maintaining the end-to-end processing latency below the query-specific bound.

2.2 Problem Statement

We now formally define the problem statement to be solved by the Load Shedder by mathematically expressing the objective function and latency constraint. A video stream V is a continuous sequence of frames $V = [f_1, f_2, \cdots, f_m]$. We define an application query as $Q = [q_1, \cdots, q_n]$, where q_i denotes a video processing operator (including filtering), which reads the output of q_{i-1} and outputs to q_{i+1}. The first operator in a query always reads the combined video stream coming from the multiple cameras it is serving. We denote the output stream of operator q_i with input stream v as $q_i(v)$. We define the set of target objects detected by a query Q in video V as $T_Q(V)$ as in Eq. (1). We represent frame f containing target object o by the relation $o \in f$.

$$T_Q(V) = \{\text{target objects in } V \text{ detected by } Q\} \tag{1}$$

Now, introducing the Load Shedder component into the video query Q, we construct a query Q' of the form $[q_0, q_1, \cdots, q_n]$, where q_0 is the Load Shedder component, also denoted by LS. The *Quality of Result (QoR)* metric we use in this work is designed to measure the number of frames for each target object that are sent downstream by the Load Shedder LS in query Q', i.e., belonging to the output stream $LS(V)$. We define the per-target-object QoR for target object o in Eq. (2). QoR metric has a value between 0 and 1.

$$QoR_Q(o, LS, V) = \frac{|f \in LS(V) : o \in f|}{|f \in V : o \in f|} \tag{2}$$

The overall QoR metric for query Q with the Load Shedder LS against source video V is calculated as the average per-object QoR metric over all target objects in V (Eq. (3)). Thus, this metric quantitatively measures the aggregate performance of the Load Shedder for a given source video.

$$QoR_Q(LS, V) = \frac{\sum_{o \in T_Q(V)} QoR_Q(o, LS, V)}{|T_Q(V)|} \tag{3}$$

For a given video stream $V = [f_1, f_2, \cdots, f_m]$, we define the end-to-end (E2E) delay experienced by frame f of video V when processed by query Q' as $E2E_{V,Q'}(f)$ which is expressed as shown in Eq. (4), where q_k is the last operator in Q' that processes frame f.

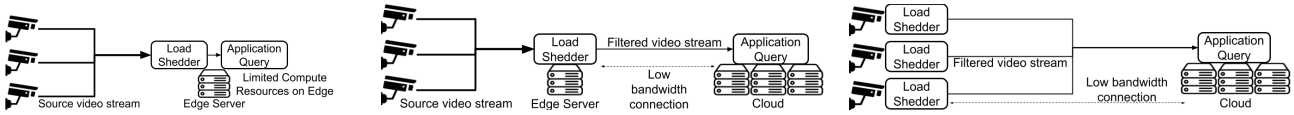
$$E2E_{V,Q'}(f) = \sum_{i=0}^{k} queue(q_i, f) + exec(q_i, f) \tag{4}$$

The objective is to maximize the Quality of Result (QoR), while dropping frames such that the end-to-end latency bound LB is met, described as follows.

$$\begin{aligned} Maximize \quad & QoR_Q(LS, V) \\ \text{s.t.} \quad & E2E_{V,Q'}(f) \leq LB \quad \forall f \in V \end{aligned} \tag{5}$$

3 RELATED WORK

Resource Management in Online Video Analytics: Live video analytics is an extremely compute and network resource intensive application. Prior art has adopted two main approaches for managing the high resource requirements of this application - tuning the configuration of the input stream and video processing operators (e.g., frame rate, resolution, etc.) and early discard of uninteresting frames. Online adaptation of camera streams and video processing operators has been explored in several prior works [16–18]. However, these works target scenarios where end-to-end latency of the order of a few seconds can be tolerated as their primary



(a) Edge server hosting the Load Shedder and application query. Compute on the edge is the bottleneck resource.

(b) Load Shedder on the Edge and application query on the cloud. Edge-Cloud network bandwidth is the bottleneck.

(c) Load Shedder on the camera and application query on the cloud. Camera-cloud network bandwidth is the bottleneck.

Figure 2: Three candidate deployment scenarios for the proposed load shedding approach.

objective is to optimize cost of resources needed for supporting all video streams while also maintaining high accuracy. Therefore, these approaches cannot be easily applied to our target application scenarios that have stringent constraints on end-to-end latency. However, their technique can be adopted to work complementary to our proposed approach.

Early-Discard Filtering of Video: Multiple works have explored early discard of unnecessary video frames to avoid the cost of streaming and processing them. Glimpse [19] proposed a hardware-software add-on that uses low-powered coarse-grained vision modalities to filter out frames irrelevant to the target query. It uses specialized hardware for motion detection, temperature measurement, etc., which are not available on cameras in our scenarios. Zhang, et al., [20] use a multi-stage pipeline of operators to determine if a frame is relevant to a query, wherein the operators are implemented using GPUs and impose significant latency overhead to the video processing pipeline. FilterForward [10] consists of a base DNN whose intermediate layers' outputs are used by a per-application binary classifier to decide whether to filter out a frame. However, in that system, the execution latency of the base model itself is very high (>200ms). These approaches do not meet our objective of imposing small overhead on the end-to-end video processing delay. The EarlyDiscard strategy proposed in Wang, et al., [21] uses an inexpensive DNN to perform filtering of frames. However, the filter cannot be tuned to ensure end-to-end video processing latency. Reducto [12] makes use of difference in low-level visual features (pixels, edges, etc.) across consecutive frames to determine if processing the new frame would result in a difference in the query's result. However, it operates at 1-second granularity and cannot guarantee end-to-end latency less than that. It also does not support tuning of the filtering based on load experienced by backend application query. EarlyDiscard and Reducto do not possess the ability to tune the frame filter based on dynamically changing workload characteristics over time, which is a key requirement in our target application scenarios to meet the end-to-end latency bound.

4 UTILITY-AWARE LOAD-SHEDDING

In this section, we present a high-level architecture of the proposed load shedding system, highlighting the interactions between different components. Then we describe the design and functionality of each individual component in isolation.

4.1 System Architecture

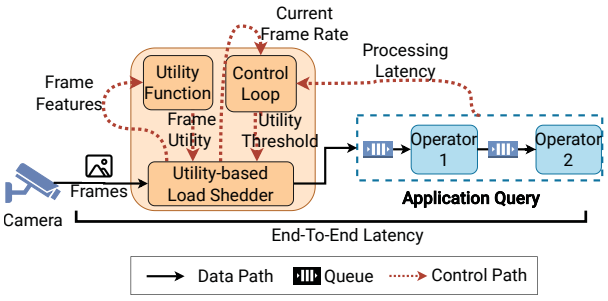


Figure 3: Proposed load-shedding architecture.

We extend real-time video analytics systems equipped with edge-computing capability (either as edge sites or co-located compute with cameras) with our proposed load shedding system that sheds a portion of the input frames to maintain the given latency bound (*LB*) during overload. Figure 3 shows the main components and their interactions—the Load Shedder, the application query, and the Control Loop. Each component (including Load Shedder) reads from an ingress queue and pushes the output data to the egress queue.

The load shedding system must perform two primary tasks—(1) decide when and how much to shed, and (2) decide which frames to shed. We describe the latter task first, for which the Load Shedder computes a utility/ importance value for each video frame that denotes the probability of the frame being *useful* for the application query. The utility is calculated using the color content of the frame and the given application query. The intuition behind the design of the Load Shedder is to discard video frames that have a low probability of being useful for the given application query (see Section 4.2). The Load Shedder can be dynamically configured with a utility threshold, such that it drops frames with utility less than the threshold.

The Control Loop component dynamically computes the utility threshold and updates the Load Shedder. The utility threshold is set such that the current load on the application query is maintained and the end-to-end latency constraint

is met (see Section 4.4). More specifically, the Control Loop component monitors the queue lengths of each operator in the query, and estimates the current observed end-to-end latency. Based on the observed end-to-end latency and the query's latency requirement, the Control Loop computes the fraction of frames that should be dropped by the Load Shedder, which is then transformed into a utility threshold (see Section 4.3). Transforming the desired frame drop rate to a utility threshold is done on the basis of the utility distribution of frames observed in the past. All of these concepts are described in the subsequent sections.

4.2 Building the utility function

The Load Shedder decides whether to discard a frame based on it's utility value - which is calculated by the utility function using the frame's color-based features. We design the utility function specifically for the target application scenarios. Firstly, since the application queries considered perform detection of objects of one or more target colors (Section 2.1.1), we use the frame's color features to calculate utility and thereby avoid missing a potential target object. Secondly, the target queries' end-to-end latency constraints necessitates the design of a light-weight load-shedder (Section 2.2). Thus the proposed utility function does not apply for queries that do not belong to the set of representative applications described in Section 2.1.1.

4.2.1 Hue-Saturation-Value Color Model. We use the Hue-Saturation-Value (HSV) model for representing the color of pixels. Hue represents the color itself (or the dominant wavelength), Saturation represents the brilliance and intensity, while Value defines the lightness or darkness. The HSV model is widely used in computer vision applications over the Red-Green-Blue (RGB) model because HSV separates the color information from intensity information (which helps deal with situations like lighting changes or shadows). The HSV triplets of all pixels represent the distribution of colors in a frame, which forms the input feature set for the utility function (described in detail in subsequent sections). The ranges of H, S and V we use are $[0, 180)$, $[0, 256)$, and $[0, 256)$ respectively. The Application Query developer specifies the color of the target objects in terms of a hue range C as input to the Load Shedder. For instance, the color red is represented using the hue ranges $[0, 10) \cup [170, 180)$.

4.2.2 Training data. To train the utility function, we use a labeled stream of videos from multiple cameras as training data set. Each element of the training data set $\mathcal{D}$ is of the form (f, l), where f represents the frame and l represents the label. Each frame f is described by a list of pixels, with each pixel being represented by a triplet (h, s, v) denoting it's Hue, Saturation and Value fields. The label l tells whether the frame contains an object of interest, i.e., whether the frame was a match for the given query. Henceforth, we use the term *positive* to denote frames which contain one or more target objects, and *negative* to denote frames that don't.

The goal of the utility function is to separate the computed utilities of positive and negative frames, such that using a utility threshold results in effective load shedding. In the rest of this section, we outline our observations from the training data on how to separate positive and negative frames using the HSV model and the construction of the utility function.

4.2.3 Hue as feature for separating positive and negative frames. We first explore the use of the Hue field of pixels to calculate whether the given frame contains a target object of the given color. We do so by computing a metric *Hue Fraction* for the color C, denoted by HF_C.

$$HF_C(f) = \frac{|\text{pixel } p \in f : hue(p) \in C|}{|\text{pixel } p \in f|} \tag{6}$$

A higher Hue Fraction would imply higher likelihood for the frame to contain a target object of a given color. Hence, a threshold-based approach on HF_C is a candidate for the Load Shedder. However, an analysis of our dataset showed that the distribution of HF_C for negative frames overlaps significantly with positive frames across all videos - as seen in Fig. 4a, which shows HF_{RED} for the entire dataset. This overlap would prevent a threshold-based approach on the hue fraction from effectively differentiating between positive and negative frames - shown in Fig. 4b, where the per-object QoR metric drops steeply with hue fraction thresholds without achieving a significant frame drop rate. We posit that this overlap in distribution is because both positive and negative frames contain red-colored pixels; however, the saturation and value fields of those pixels in the positive and negative frames would have distinctly discernible distributions.

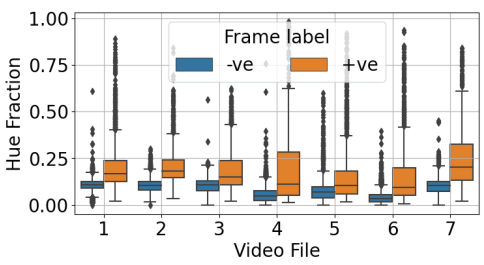
4.2.4 Using Saturation and Value fields for differentiating frames. In order to separate positive frames from negative ones, we analyze the distribution of Saturation and Value fields for Red pixels across videos. We discretize the range of saturation and value into bins of size s and v respectively using functions sat_bin and val_bin that map a pixel's saturation $sat(p)$ and value $val(p)$ to their corresponding bins.

$$sat\_bin(p) = i \iff i \cdot s \leq sat(p) < (i+1) \cdot s \tag{7}$$

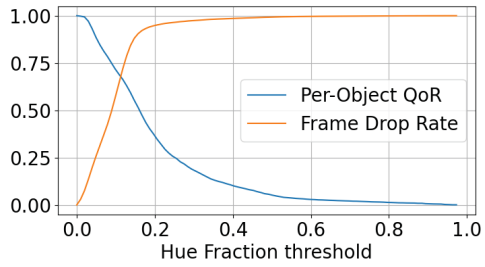
$$val\_bin(p) = j \iff j \cdot v \leq val(p) < (j+1) \cdot v \tag{8}$$

We now transform a frame f into a 2D matrix for a given color C, i.e., $PF_C^{(i,j)}$, so that the matrix element $PF_C^{(i,j)}$ denotes the fraction of pixels whose hue falls in the range C and their saturation and value fall into the i^{th} and j^{th} bin respectively.

$$PF_C^{(i,j)}(f) = \frac{|\text{pixel } p \in f : hue(p) \in C \wedge in\_bin(p, i, j)|}{|\text{pixel } p \in f : hue(p) \in C|} \tag{9}$$



(a) Hue Fraction of color RED across training videos.



(b) QoR and Drop Rate vs. HF threshold for RED color.

Figure 4: Using Hue Fraction for selecting positive vs. negative frames. Fig. 4a shows that Hue Fraction of negative and positive frames has significant overlap. Fig. 4b shows that high frame drop rate results in a steep drop in QoR.

$$in\_bin(p,i,j) = (bin_{sat}(p) = i \wedge bin_{val}(p) = j) \tag{10}$$

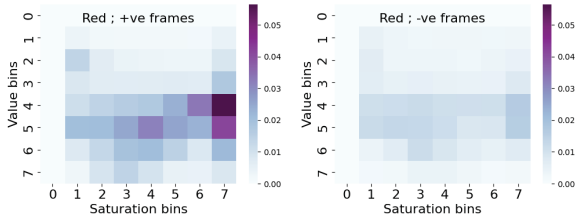


Figure 5: Distribution of Saturation and Value fields for positive ($M_{C,true}$) and negative frames ($M_{C,false}$) across all videos in the dataset. Bins with high saturation are better differentiators of positive frames.

Using the per-frame distribution of pixels in saturation-value bins, we quantify how useful each bin is in classifying a frame as positive or negative. We compute a metric for each saturation-value bin using the pixel distribution of positive and negative frames that denotes the correlation of the bin with the particular label (+ve or -ve) for a given frame.

$$M^{(i,j)}_{C,+ve} = \text{AVG}\ PF^{(i,j)}_{C}(f)\ \forall\ (f,1) \in \mathcal{D} \tag{11}$$

$$M^{(i,j)}_{C,-ve} = \text{AVG}\ PF^{(i,j)}_{C}(f)\ \forall\ (f,0) \in \mathcal{D} \tag{12}$$

The distribution of above utility for the color Red, i.e., $M_{C,+ve}$ and $M_{C,-ve}$ is shown in Fig. 5. Bins with higher saturation are stronger indicators of whether a frame is positive.

4.2.5 Computing per-frame utility. We use the utility of each saturation-value bin (from Eq. (11) and Eq. (12)) to compute the utility value for a frame. The utility is a weighted sum of the $M^{(i,j)}_{C,+ve}$ value for each saturation-value bin, weighted by the pixel fraction of the frame in that bin.

$$U_C(f) = M^{(i,j)}_{C,+ve} \cdot PF^{(i,j)}_{C}(f) \tag{13}$$

Fig. 6 schematically describes the approach of building the utility function using training data and using the function to calculate utility value for frames at runtime.

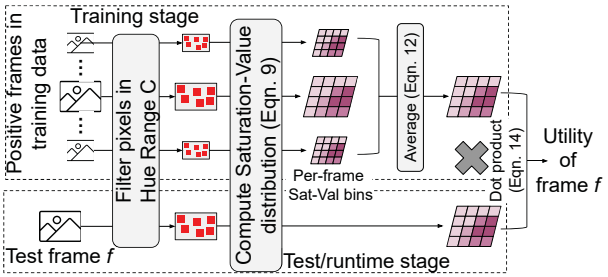


Figure 6: The proposed utility-based load shedding approach - with both training and testing/runtime stages.

4.2.6 Computing per-frame utility for composite color queries. Computing the utility of a frame for composite queries requires using the utility function for the component colors.

$$U_{C_1 \vee C_2}(f) = max\left(\overline{U_{C_1}}(f), \overline{U_{C_2}}(f)\right) \tag{14}$$

$\overline{U_{C_1}}$ represents the utility function for color C_1 normalized over the training data, such that the maximum utility is 1.0. Normalization of per-color utility functions allows their effective composition. Similarly for computing the utility of a query looking for both colors C_1 **AND** C_2 in a frame, we use the minimum of $\overline{U_{C_1}}$ and $\overline{U_{C_2}}$ as the composite utility.

4.3 Computing Utility Threshold

The Load Shedder's utility threshold at any point of time is computed using the current target frame drop rate - based on the distribution of utility values for a set of recent frames. Using the utility function U_C, we build a cumulative distribution function (CDF) of utility values over frames in the history $\mathcal{H}$, as shown in Eq. (15).

$$CDF(u) = \frac{|\{f : U_C(f) \leq u\ \forall\ (f,l) \in \mathcal{H}\}|}{|\mathcal{H}|} \tag{15}$$

A CDF-based approach allows incorporating utility values of more recent frames into $\mathcal{H}$ to update the CDF with changing video content. Initially, the training set $\mathcal{D}$ itself can be used as the set $\mathcal{H}$. To determine the utility threshold for a given target drop rate r, we use the inverse of the CDF function iteratively to compute the minimum utility value u_{th} such

that $CDF\,(u_{th}) \geq r$. The intuition behind this approach is that for the utility threshold u_{th}, the Load Shedder will drop a fraction r from $\mathcal{H}$. Since the utility distribution of recent historical frames is expected to be similar to new frames in the near future, the Load Shedder is expected to drop r fraction of new incoming frames. However, the observed frame drop rate of new incoming frames might not equal the target drop rate r because it's transformation to utility threshold depends entirely on the distribution of frame utilities in $\mathcal{H}$.

4.4 Design of Control Loop

The utility and threshold calculation in Section 4.1 are used by a Load Shedder to control the end-to-end (E2E) latency of execution of a video processing query. The Load Shedder requires a control loop (Fig. 3) to define the target drop rate to keep the E2E latency within the specified bound.

4.4.1 Control Loop mechanisms. The developer of the video query defines the required E2E latency, which guides the execution of the Load Shedder. Besides this requirement, the Load Shedder uses as inputs both the frames per second (FPS), the (current) processing latency of backend query execution, and the (current) network latency between camera and Load Shedder, and between Load Shedder and the backend running the query. The Load Shedder uses two main mechanisms to control the end-to-end latency: admission control and dynamic queue sizing. These mechanisms differ primarily in how quickly they adapt the target drop rate based on changes in backend execution load.

Admission Control. It decides which frames are considered for further processing. It monitors the per-frame execution and queuing delays of a frame for all operators of the Application Query and computes the average perceived query processing latency ($proc_Q$). The currently supported throughput (ST) by the backend query is then calculated as: $ST = \frac{1}{proc_Q}$. The ST is then compared against the FPS (frame per second) coming into the Load Shedder to calculate the required target drop rate as follows:

$$Target\ Drop\ Rate = max(0, 1 - \frac{ST}{FPS}), \tag{16}$$

which prescribes the frame drop rate to match the frame rate ingress into the backend query with the available processing throughput, such that the system is stable. As described in Section 4.3, we transform the target drop rate to a utility threshold by using the utility distribution of historical frames, to filter new ingress frames based on their utility value.

Dynamic Queue Sizing. Internally, the Load Shedder (as shown in Fig. 3) also manages a queue that it uses to ensure that the the end-to-end latency of any frame accepted by admission control is met. The expected E2E latency for the N^{th} frame in the Load Shedder queue is shown in Eq. (17).

$net_{cam,LS}$ and $net_{LS,Q}$ represent the average of continuously monitored network latencies between cameras and Load Shedder and between Load Shedder and query backend respectively. $proc_{CAM}$ denotes the average latency incurred in processing frames on the camera (including background subtraction, feature extraction, etc.) (analyzed in Section 5.6).

$$\begin{aligned} Expected\ E2E\ Latency = (N+1) \cdot proc_Q + \\ net_{cam,LS} + net_{LS,Q} + proc_{CAM} \end{aligned} \tag{17}$$

If one of the component latencies increases, later frames in the Load Shedder's internal queue could violate the E2E latency requirement. Dynamic queue sizing helps reduce the likelihood of latency violation for a frame with high utility.

Dynamic queue sizing updates the size of the Load Shedder's queue and drops the lowest utility frames if needed, allowing frames with the highest utilities to be processed. Dynamic queue sizing reacts faster than updates to the utility threshold (that guides admission control) and reduces the likelihood of an E2E latency violation. The queue is always at least of size one to avoid starving the downstream operators.

Dynamic queue sizing can also be seen as a second layer of admission control. Even if a new frame has a utility higher than the current threshold, it will be dropped if the queue is full and it has the lowest utility of all frames currently in the queue. Similarly, if an incoming new frame has a greater utility than the lowest utility frame that is already in the queue, then the latter will be dropped and the new frame added to the queue. This queue shedding keeps the latency requirement valid even for new incoming frames.

Both mechanisms allow the Load Shedder to fine-tune the admission of new frames to extract the most utility out of the video stream while maintaining the required E2E latency.

5 EVALUATIONS

In this section, we present the results of experimental evaluations carried out to test the efficacy of the proposed utility-based load shedding approach. Our experiments are tailored to validate the following hypotheses.

(1) The proposed utility function can compute utility value for unseen video frames (not in training set) and effectively differentiate between positive and negative frames.
(2) The proposed control loop adapts to changing workload pattern and is able to meet application performance requirements without sacrificing QoR metric.
(3) The utility value calculation is light-weight and imposes low overhead on edge devices.

5.1 Data Set

We generated a benchmark of synthetic videos with Visual-Road [22], which is a benchmark to evaluate video database management systems. VisualRoad uses the autonomous driving CARLA simulator [23] to generate videos from CCTV

cameras located in a realistic city-like environment, including pedestrians, bicycles, different types of vehicles, and all the surroundings (roads and buildings). Additionally, it allows perturbing the locations of cameras (by specifying a seed parameter) and weather conditions, thereby generating a number of different scenarios.

We evaluate our proposed color-based Load Shedder and associated control loop with 25 videos from 7 seeds value (3 or 4 videos from each seed value) using sunny weather. Each video represents 15 minutes of a camera video stream facing a road or highway in a city, with a frame rate of 10 fps. Different cameras have different distributions of cars, varying from cars always presents to rarely appearing. In our results, we report metrics for videos that contained a decent number of target objects for the given query.

5.2 Implementation

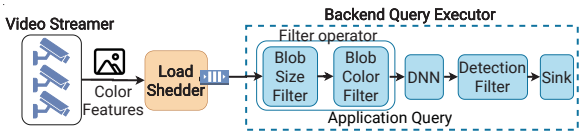


Figure 7: The components used in the evaluation setup.

To evaluate the aforementioned hypotheses, we implement the video processing system along with the Load Shedder. It has three main components: the Video Streamer, the Load Shedder, and the Backend Query executor, as shown in Fig. 7. All the components exchange messages using the communication library ZeroMQ [24], with the messages serialized using Cap'n Proto [25]. The Load Shedder is implemented in Python 3 and all the other components in C++. The Video Streamer reads the video files generated with VisualRoad, performs background subtraction, extracts the color features for each frame (as described in Fig. 6) and streams them to the Load Shedder. The Video Streamer component is capable of emulating multiple cameras sending their frames' features to the Load Shedder by interleaving their frames. Next, the Load Shedder implements the utility calculation and load shedding described in Section 4.1. The utility function we use in our evaluations uses 8 bins for both saturation and value, meaning that the bin sizes s and v are equal to 32. Through preliminary experiments (not shown in this paper) we found that these bin sizes offer the best separation of positive from negative frames. In the real-world the Load Shedder would be fed raw video frames from multiple cameras, instead of from the Video Streamer. Since the Load Shedder consumes raw frames, a real-world deployment would need video streams to be pre-processed by a video decoder and decompressor before being consumed by the Load Shedder. The proposed Load Shedder's implementation does not impose any constraint on the input camera streams' characteristics such as frame rate, encoding or compression techniques. Finally, the Backend Query Executor runs the video analytics query to be executed on the video stream. The Load Shedder and Backend Query Executor run on a NC6 Virtual Machine on Microsoft Azure with 6 Intel Xeon E5-2690 v3 vCPUs, 56 GiB of RAM and an NVIDIA Tesla K80 GPU.

There are two additional components to implement the control loop: the Metrics Collector and a Transmission Control Mechanism. The Metrics Collector measures the total incoming frame rate and the end-to-end latency in the Backend Query Executor, and forwards these metrics to the Load Shedder. The Load Shedder uses this information to define the target drop rate. The Transmission Control Mechanism implements a token-based backpressure algorithm between the Load Shedder and the Backend Query Executor - wherein in the event of low backend query load, the Load Shedder sends more frames to the backend query, while under high backend query load, the Load Shedder follows the utility threshold and drops low-utility frames.

5.3 Video queries

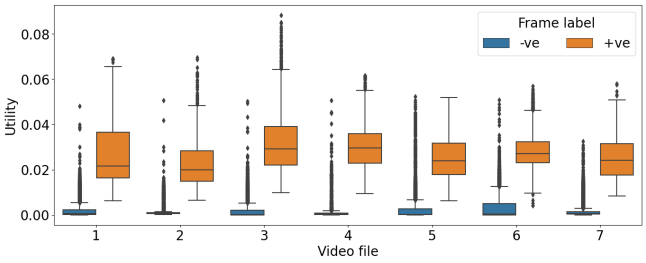
We consider object detection and tracking queries that need to look at multiple frames of target objects. The model query we use consists of (1) a filter component that groups together spatially adjacent pixels into blobs and drops frames that do not have at least one blob of a certain minimum size, (2) a second filter that ignores frames that do not have a blob(s) of the target object's color, (3) a DNN that performs object detection, (4) a filter that looks for the detected objects' color and label before sending the information to the sink. We use the *efficientdet-d4* [26] object detection DNN. We evaluate simple queries for target objects of a single color and composite queries for objects of multiple colors.

5.4 Performance on Unseen Videos

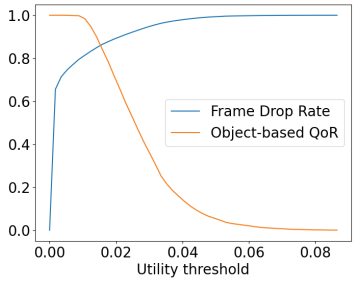
We evaluate the performance of the Load Shedder on unseen videos using an iterative cross-validation study. We split the video dataset into training and testing set, build the Load Shedder's utility function using the training set and compute the utility and correctness metrics for the test videos.

5.4.1 Single-color query: Red. We first evaluate a query for detecting target objects of a single color, i.e., red. Fig. 8a shows that the utility of positive frames is significantly higher than the negative ones in our dataset, thereby demonstrating the efficacy of the utility function on unseen videos.

We demonstrate that such a higher utility for positive frames helps in detecting target objects and maintaining a high QoR value while also shedding a significant fraction of (useless) frames. In Fig. 8b, we show how an increasing utility threshold causes higher frame drop rate, which also



(a) Utility values for positive and negative frames of unseen video frames. For a given video in the figure, the utility function used to compute its frames' utility values is trained using data that does not contain the given video.



(b) Target object based QoR metric and frame drop rate vs. utility threshold for a Load Shedder tuned to detect red vehicles.

Figure 8: Performance of the utility-based Load Shedder on a query looking for Red cars as target objects.

includes a small portion of useful frames containing target objects, and hence results in a drop in the QoR metric.

Comparison against Content-agnostic load shedding. We compare the performance of the proposed utility-based load shedding approach against a content-agnostic approach that sheds a fixed rate of incoming frames using a uniform probability. Firstly, Fig. 9a shows the variation of frame drop rate and the per-object QoR against the target drop rate of the Load Shedder. Even at a high frame drop rate, the QoR remains at 1.0 due to frequent low-utility negative frames. The QoR drops only when the target drop rate becomes so high that higher-utility frames (containing target objects) need to be dropped. Similarly, Fig. 9b shows the frame drop rate and QoR against target drop rate for the *Content-agnostic* shedding approach (with each setting repeated 20 times). With increasing target drop rate, the QoR falls sharply because content-agnostic shedding often sheds frames containing target objects. Fig. 9c compares the QoR that the proposed load shedding approach can achieve for a given observed frame drop rate with the content-agnostic approach. Unlike continuous decline of QoR for the content-agnostic approach, the QoR for utility-based approach has a visible drop only when the observed frame drop rate gets close to 1.0. The result shows that the utility-based approach is selective in picking frames to send to Backend Query Executor, and can achieve a much higher QoR for a given observed frame drop rate compared to a content-agnostic shedding approach.

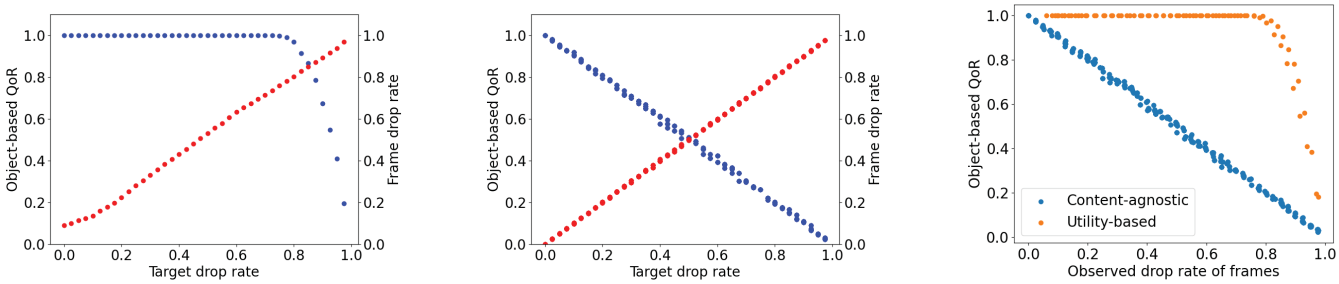
5.4.2 Composite-color query: Red OR Yellow. We perform a similar analysis for two composite queries - (1) detect target objects that are either Red OR Yellow in color, and (2) detect all frames containing both Red AND Yellow target objects. As before, we iteratively select a set of videos as the training set and the complementary set as the test set. The utility value of frames for the OR query is shown in Fig. 10a. As for the single-color query, positive frames have significantly higher utility than negative frames. Note that for the composite OR query, a positive frame is one that contains either a Red or a Yellow target car. Fig. 10b shows the frame drop rate and QoR against the utility threshold. The QoR remains stagnant at 1.0 (selecting all frames containing target objects) with a high frame drop rate, until the utility threshold becomes high enough to start dropping positive frames. Fig. 11 shows the utility value of frames for the AND query, and the differentiation between positive and negative frames is visible here as well. Note that for the composite AND query, a positive frame should contain both a Red and a Yellow car.

5.5 Application Evaluations

In this subsection, we detail an E2E evaluation of running both the utility calculation and the Load Shedder control loop using a real-time video stream and show how it controls the frame processing latency and avoids backend overload.

5.5.1 Synthetic scenario. First, we evaluate a synthetic worst-case scenario in which a sudden burst of high-processing activity occurs in the ingress video. The video comprises three segments: low-utility frames with no target object, high-utility frames containing target object(s), and high-utility frames with no target object. To create such a tailored video, we obtain segments from the videos generated with Visual Road that are known a-priori to have those properties, and stitch them together to form a 15 minutes long video with each of the above three segments being 5 minutes long.

We expect that during the video's first low-utility no-object segment, the Load Shedder will allow frames to be processed by the filter-stage in the backend query, despite low frame utility. This is because the filter operator would drop these frames as they don't contain target objects anyway. Hence, the processing latency $proc_Q$ is low, and a low target drop rate is set (Eq. (16)). In the second segment of the video, the Load Shedder starts shedding frames because all frames are be processed by the expensive DNN as they contain target objects. Thus, the Load Shedder increases the utility threshold so that the backend query executor can keep the end-to-end latency bounded. Finally, in the third segment

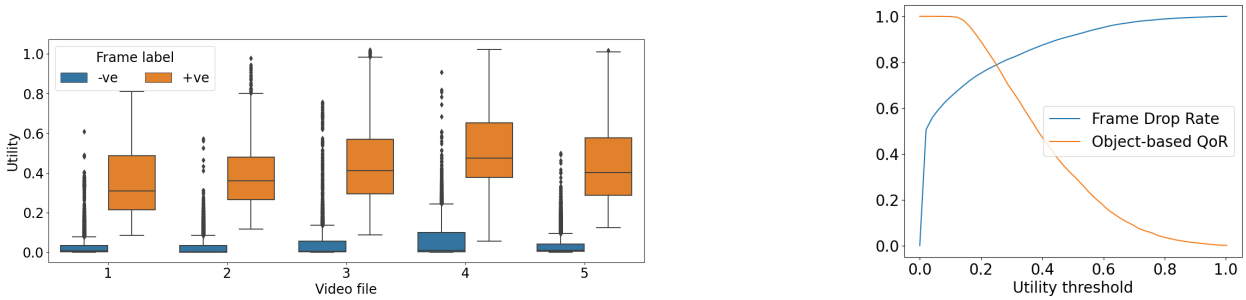


(a) Impact of target drop rate on QoR rate and frame drop rate for utility-based approach.

(b) Impact of target drop rate on QoR and frame drop rate for content-agnostic approach.

(c) QoR vs. frame drop rate trade-off for utility-based and content-agnostic load shedding approaches.

Figure 9: Fig. 9a and Fig. 9b show the impact of target drop rate on the QoR metric and observed frame drop rate for the proposed utility-based and the content-agnostic probabilistic approach respectively. Tradeoffs between the two aforementioned metrics are shown for both approaches in Fig. 9c.



(a) Utility values for positive and negative frames of unseen video frames. For a given video in the figure, the utility function used to compute its frames' utility values is trained using data that does not contain the given video.

(b) Target object detection rate and frame drop rate vs. utility threshold for a Load Shedder tuned to detect red or yellow cars.

Figure 10: Performance of utility-based Load Shedder on query looking for Red OR Yellow cars as target objects.

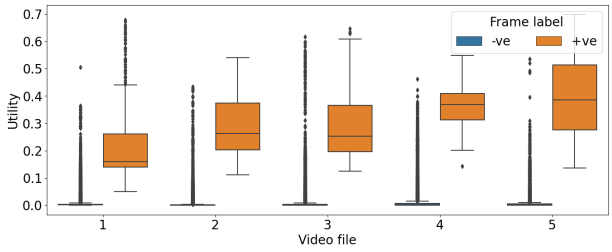


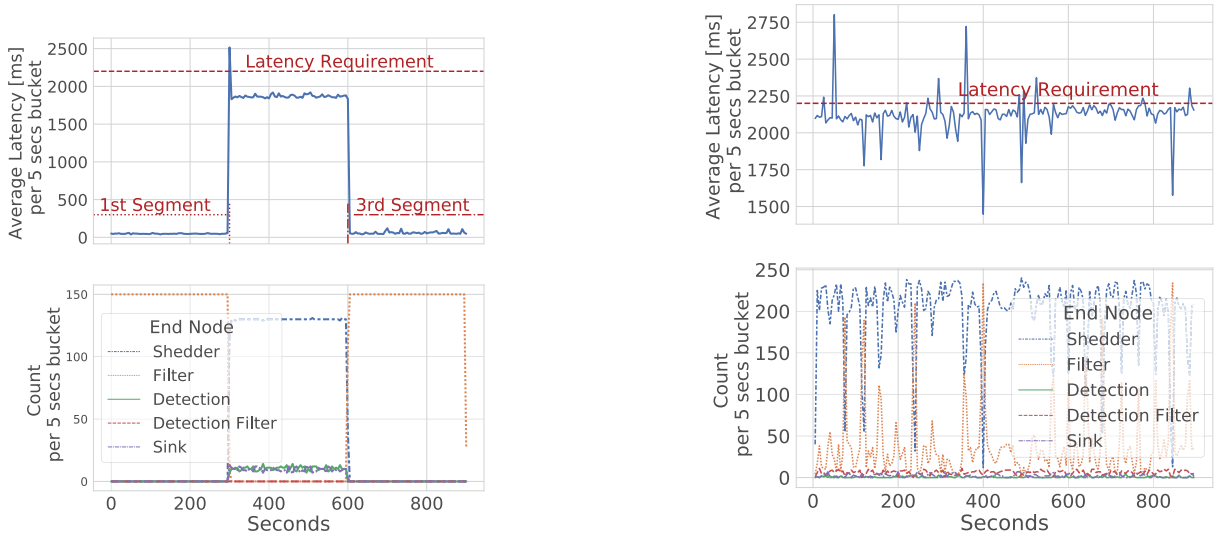
Figure 11: Utility of unseen video frames for a composite query that detects Red and Yellow cars.

with no target objects, the Load Shedder stops shedding again and its execution profile resembles the first segment.

Fig. 12a shows the time-varying behavior of the query execution. The upper graph of Fig. 12a shows the max E2E latency for each 5 minute time window, along with the E2E requirement. The lower graph in Fig. 12a shows the number of frames processed at each stage group every 5 seconds. The upper graph in Fig. 12a shows that the latency is always less than the E2E latency bound. Similarly, the lower graph in Fig. 12a shows that the Load Shedder decides the frame drop rate at each segment, with no shedding in the 1st and 3rd segments and plenty of shedding in the 2nd segment. Both these results show that the proposed Load Shedder can quickly react to changes and few violations even with extreme changes, that too for real-time video processing with 24 FPS input frame rate. There was only 1 latency violation during the peak in the 2nd segment while the Load Shedder was recalculating the queue size and the utility threshold.

5.5.2 Realistic smart-city scenario. We analyze the Load Shedder running directly on videos generated using Visual Road. The Video Streamer emulates workload from multiple cameras by generating a stream of frame features interleaved from multiple videos. We show the E2E latency and the distribution of frames processed at each stage, and the variation of the QoR metric with the number of concurrent video streams.

As in Fig. 12a, Fig. 12b shows that the Load Shedder can bound the processing latency by dropping frames. The synthetic scenario reports more spikes because the DNN is invoked unpredictably by frames from different videos at different times. The Load Shedder minimizes latency violations caused due to these sudden load surges. Fig. 12b shows the latency and elements being processed for five concurrent



(a) Synthetic Scenario each segments highlighted

(b) Realistic Scenario

Figure 12: Analysis of both a synthetic and a realistic scenario. The upper graph of both figures shows the avg. processing latency on the y-axis and video time on the x-axis, along with the end-to-end latency requirement. The lower graph shows the number of frames that reached each query component (from Fig. 7) over video time.

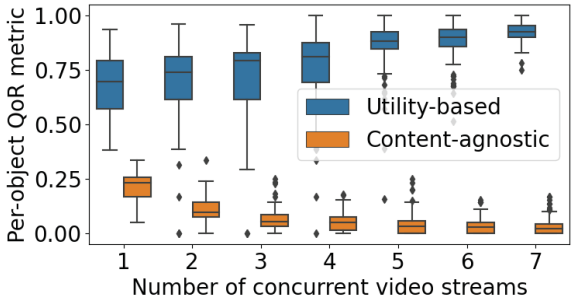


Figure 13: QoR of the proposed utility-based vs. content-agnostic approach with varying number of concurrent video streams.

videos. Fig. 13 shows that the proposed utility-based approach can exploit the statistical multiplexing between multiple cameras' video streams and achieve a high QoR. On the other hand, a content-agnostic approach has poor QoR. We compute the target drop ratio for the latter approach assuming that $proc_Q$ is 500 ms, which is a rather lenient assumption for the baseline.

5.6 Runtime Overhead Analysis

We evaluate the additional latency incurred by a video analytics system using the proposed Load Shedder vs. executing the backend query without load shedding. We assume that cameras have colocated compute capability, which is used for: (1) converting color space from RGB to HSV, (2) background subtraction, and (3) color feature extraction. We evaluate the time taken to perform these tasks on an Nvidia Jetson TX1 with a quad-core ARM Cortex-A57 processor and 4 GB of RAM (representative of compute power co-located with cameras). We use a video stream with continuously high activity to stress the system and obtain worst-case latency numbers. Table 1 shows the median latency incurred by each component task. The overall latency overhead remains below 35 ms which can support video streams of multiple cameras operating at 10 FPS or higher. For composite queries the feature extraction latency is multiplied by the number of colors involved, while the other components' latencies would not change (as they are computed once per frame).

Component	Median Latency Overhead
Color Space Transformation	3.5 ms
Background Subtraction	26.4 ms
Feature Extraction	3.8 ms

Table 1: Breakdown of Load Shedder latency overhead. The utility calculation component is not shown because of its negligible latency.

6 DISCUSSION

Automatic selection of Hue ranges for a query. The proposed load shedding approach requires minimal intervention of the application query developer, except having to provide the Hue range for the target objects. This task can be automated by analyzing bounding boxes of target objects in the training data. Techniques such as dominant color detection [27] can be used to automatically extract the Hue ranges in target objects and fed into the utility calculation function.

Feature calculation vs utility calculation on camera.

Network efficiency can further be improved by pushing the utility calculation itself to the camera. This involves a trade-off between the overhead of maintaining a distributed utility model versus a higher communication cost of sending additional low-utility frames. When cameras can calculate utility, the load shedder can tune their utility threshold to reduce unnecessary frames sent. However, the utility model needs to be updated at each camera, incurring additional bandwidth. Therefore, this decision should be taken considering the connectivity scenario of the camera network.

7 CONCLUSION

In this paper, we present a low-cost Load Shedder for real-time video processing on edge devices which sheds frames such that the per-frame end-to-end processing latency is within the bound for the query, while maximizing the quality of result (QoR). The Load Shedder uses color features of ingress video frames to compute a utility value that denotes the probability of a given frame containing a target object of the query. We propose a utility threshold based approach for the Load Shedder to enforce a target frame drop rate. The Load Shedder consists of a control-loop component that continuously monitors execution and detects overload in the backend query. Through evaluations we have shown that the proposed Load Shedder is able to differentiate between frames containing target objects from those that don't. It is able to attain a high frame drop rate while maintaining a high QoR metric. Additionally, the Load Shedder is able to adjust the utility threshold dynamically based on the observed load on the backend query to meet the end-to-end latency bound. Finally, we show that the overhead of the load shedding approach is not significant on edge devices.

ACKNOWLEDGEMENT

This work was supported by the German Research Foundation (DFG) under the research grant "PRECEPT II" (BH 154/1-2 and RO 1086/19-2).

REFERENCES

[1] Z. Xu, S. Sinha, S. Harshil S, and U. Ramachandran, "Space-time vehicle tracking at the edge of the network," in *Proc. of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*. ACM, 2019.

[2] T. T. Le, S. T. Tran, S. Mita, and T. D. Nguyen, "Real time traffic sign detection using color and shape-based features," in *Asian Conference on Intelligent Information and Database Systems*. Springer, 2010.

[3] Wobot.ai. (2021) Team Wobot how is video analytics driving industry 4.0. Wobot Intelligence. [Online]. Available: https://wobot.ai/video-analytics/how-is-video-analytics-driving-industry-4-0/

[4] 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.

[5] U. Ramachandran, H. Gupta, A. Hall, E. Saurez, and Z. Xu, "A case for elevating the edge to be a peer of the cloud," *GetMobile: Mobile Computing and Communications*, vol. 24, no. 3, pp. 14–19, 2021.

[6] M. Zhang, F. Wang, Y. Zhu, J. Liu, and Z. Wang, "Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines," in *Proc. of the ACM Multimedia Systems Conference*, 2021.

[7] N. Tatbul and S. Zdonik, "Window-aware load shedding for aggregation queries over data streams," in *Proc. of VLDB*, 2006.

[8] A. Slo, S. Bhowmik, and K. Rothermel, "espice: Probabilistic load shedding from input event streams in complex event processing," in *Proc. of the 20th International Middleware Conference*, 2019.

[9] A. Slo, S. Bhowmik, A. Flaig, and K. Rothermel, "pspice: partial match shedding for complex event processing," in *Proc. of the Int. Conf. on Big Data (Big Data)*. IEEE, 2019.

[10] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *CoRR*, vol. abs/1905.13536, 2019.

[11] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao, "A fast filtering mechanism to improve efficiency of large-scale video analytics," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 914–928, 2020.

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

[13] U. K. Pillai and D. Valles, "Vehicle type and color classification and detection for amber and silver alert emergencies using machine learning," in *IEEE Int. IOT, Electronics and Mechatronics Conference*, 2020.

[14] S. Roy and M. S. Rahman, "Emergency vehicle detection on heavy traffic road from cctv footage using deep convolutional neural network," in *Proc. of IEEE ECCE*, 2019.

[15] L. A. Varga, B. Kiefer, M. Messmer, and A. Zell, "Seadronessee: A maritime benchmark for detecting humans in open water," in *Proc. of the IEEE/CVF Conference on Applications of Computer Vision*, 2022.

[16] 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. of 14th {USENIX} NSDI*, 2017.

[17] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proc. of the Conf. of the ACM Special Interest Group on Data Communication*, 2018.

[18] 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 *IEEE/ACM 29th Int. Symp. on Quality of Service*, 2021.

[19] S. Naderiparizi, P. Zhang, M. Philipose, B. Priyantha, J. Liu, and D. Ganesan, "Glimpse: A programmable early-discard camera architecture for continuous mobile vision," in *Proc. of the 15th Annual Int. Conf. on Mobile Systems, Applications, and Services*. ACM, 2017.

[20] C. Zhang, Q. Cao, H. Jiang, W. Zhang, J. Li, and J. Yao, "A fast filtering mechanism to improve efficiency of large-scale video analytics," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 914–928, 2020.

[21] 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 *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.

[22] B. Haynes, A. Mazumdar, M. Balazinska, L. Ceze, and A. Cheung, "Visual road: A video data management benchmark," in *SIGMOD*, 2019.

[23] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conf. on robot learning*, 2017.

[24] ZeroMQ, "Zeromq," https://zeromq.org/, 2022, [Online; accessed 2022-01-14].

[25] Cap'n Proto, "Cap'n proto: Serialization protocol," https://capnproto.org/, 2022, [Online; accessed 2022-01-14].

[26] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proc. of the IEEE/CVF conference on computer vision and pattern recognition*, 2020.

[27] PyPi, "Dominant color detection," https://pypi.org/project/dominant-color-detection/, 2020, [Online; accessed 2022-01-31].