VU: Edge Computing-Enabled Video Usefulness Detection and Its Application in Large-Scale Video Surveillance Systems

Hui Sun, Weisong Shi, *Fellow, IEEE*, Xu Liang, and Ying Yu

***Abstract*—In the era of smart and connected communities, video surveillance systems, which typically involve tens to thousands of cameras, have increasingly become prominent components for public safety. In current practice, when a failure occurs in a video surveillance system, the operation and maintenance teams usually spend a substantial amount of time locating and identifying the failure; hence, the fast online response cannot be guaranteed in a large-scale video surveillance system. Meanwhile, the video data that contains potential failures consumes bandwidth that could be used for useful video data. The useless video will waste the scarce bandwidth in the network and storage usage in the cloud. The emergence of edge computing is highly promising in video preprocessing with an edge camera. A video surveillance system is a killer application for edge computing. In this article, we propose an edge computing-enabled video usefulness (i.e., VU) model for large-scale video surveillance systems. We also explore its application, e.g., early failure detection and bandwidth improvement. According to the usefulness of the video data, the VU model can locate a failure and send it to end-users on the fly. In this article, our goals are threefold: 1) proposing a comprehensive VU model. To the best of our knowledge, this is the first work to explore the feasibility of the VU model and to determine VU values in a real application; 2) reducing the mean time to detection (i.e., MTTD) efficiently via edge computing-enabled fast online failure detection approaches; and 3) relieving the network bandwidth for large-scale video surveillance systems. Our experimental results demonstrate the approaches in VU model accurately detect failures that were collected from a video surveillance system with approximately 4000 cameras. The MTTD is substantially shortened by the fast online detection approaches. The video data with the worst VU values is mostly discarded to lessen overload of the network.**

***Index Terms*—Cloud computing, edge computing, video surveillance systems, video usefulness (VU).**

Manuscript received February 16, 2019; revised June 6, 2019, June 25, 2019, and July 25, 2019; accepted August 12, 2019. Date of publication August 20, 2019; date of current version February 11, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61702004, in part by the Key Technology Research and Development Program of Anhui Province under Grant 1704d0802193, in part by the Natural Science Foundation of Anhui Province under Grant 1708085QF160, in part by the Natural Science Research Projects at Higher Institutions in Anhui Province under Grant KJ2017A015, and in part by the Anhui University Funding for Doctoral Research under Grant J01003214. *(Corresponding author: Weisong Shi.)*

H. Sun, X. Liang, and Y. Yu are with the School of Computer and Science, Anhui University, Hefei 230012, China (e-mail: sunhui@ahu.edu.cn; ahu_lx@163.com; yingyu@stu.ahu.edu.cn).

W. Shi is with the Department of Computer Science, Wayne State University, Detroit, MI 48202 USA (e-mail: weisong@wayne.edu).

Digital Object Identifier 10.1109/JIOT.2019.2936504

I. Introduction

With the expansion of a city's scale, public safety is essential for urban stability [1], [2]. Numerous video surveillance systems [3] have been ubiquitously deployed throughout cities and surroundings, e.g., streets, office buildings, etc. These linked devices are essential to the public security for a large-scale city's ecosystem. For example, a large-scale city such as Beijing or London has approximately one million cameras deployed [4]. In a large-scale video surveillance system, a key challenge is the elimination of useless video on the fly to store high-quality video data from cameras while maintaining low data storage usage.

A camera captures huge amounts of video data and sends it to the cloud nonstop in 24/7 mode. The compression and encoder techniques [5], [6] are used to preprocess video data. Intelligent approaches (e.g., content analysis [7], moving object detection [8], and action analysis [9]) are utilized in the cloud. According to a study by Seagate Technology LLC [10], 566 petabytes (PB) of data is generated each day by new video surveillance cameras that are installed worldwide in 2016. This daily data volume is expected to reach 3500 PB by 2023.

How to efficiently manage the large-scale video surveillance system if a failure occurs in one of its components is a primary challenge. Addressing the challenge is essential to improve the efficiency of large-scale video data in the best way.

In a video surveillance system, many failures that impact the video usefulness (VU) cannot be timely detected. Large staffs are employed to monitor and detect these failures, such as quality of service (i.e., QoS) [11] and quality of user experience (i.e., QoE) [12] failures from video streams. This failure detection approach is error-prone and expensive. In addition, it prolongs the mean time to detection (MTTD) [13].[1] A large workforce is expensive but still cannot realize fast online detection goals and has an unacceptably large MTTD. A camera with a failure produces video data that may be useless and is uninterruptedly uploaded to the cloud, which aggravates the overload in the network and wastes storage usage in the cloud before the failure is detected.

A. Problem Statement

In this article, we intend to answer the following questions.

[1]The MTTD is defined as the average latency in detecting a failure in a video stream in this article.

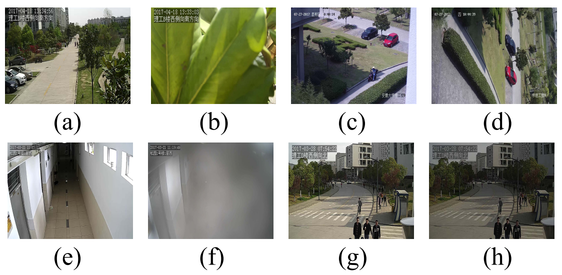


Fig. 1. Failures in a video stream. Normal video versus videos with four types of failures: 1) normal (a) versus occlusion failure (b); 2) normal (c) versus tilt failure (d); 3) normal (e) versus blurring failure (f); and 4) normal (g) versus brightness failure (h).

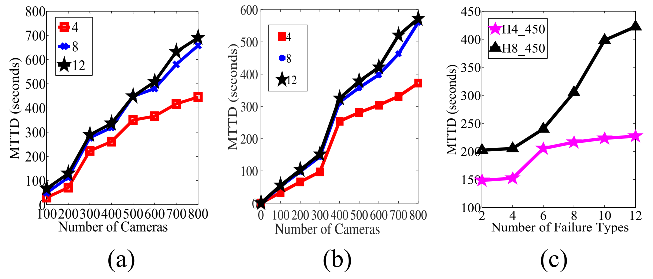


Fig. 2. MTTDs of 4, 8, and 12 failures for cameras with (a) H8-hop and (b) H4-hop network distances. Fig. 2(c) presents the MTTDs of 12 failures for 450 cameras with the H4-hop (H4_450 for example) and H8-hop network distances.

1) What is the state-of-the-art failure status in the large-scale video surveillance systems?
2) How can a failure in the video data be detected on the fly to avoid incurring the cost of storing useless video data?
3) How can these failure detection approaches be deployed using the edge computing model?

We study the video content in a video surveillance system at Anhui University. In Fig. 1(b), video data with an occlusion failure exhibits satisfactory QoS metrics in the network without failures (e.g., delay and packet loss); however, the video content is useless for video analytics. It is inefficient for end-users to identify the occlusion failure in video data. The video data with no error in the QoS will be uploaded to the cloud; however, this will waste network bandwidth and storage usage in the cloud. Video data with a blurring failure [see Fig. 1(f)] exhibits a poor QoE, which interferes with the users' decisions regarding its content and behavior. Useless video data aggravates overload in the network.

To identify possible failures in a large-scale surveillance system, we leverage a commercial detection system that is based on the cloud computing model to study MTTD for failures in the video surveillance system of Anhui University. The system can detect up to 12 types of failures.[2] Without loss of generality, we measure the MTTDs of four, eight, and 12 types of failures using the 800 sampled cameras in the surveillance system. Two types of network distance from the cloud to a camera are considered: the H4- and H8-hop network distances. The notation H4 indicates that the number of hops from the cloud to a camera satisfies $1 \leq \text{hops} \leq 4$, while the notation H8 indicates $4 < \text{hops} \leq 8$ is satisfied.

In Fig. 2, the MTTDs for detecting four, eight, and 12 types of failures from 800 cameras with H8 [see Fig. 2(a)] and H4 [see Fig. 2(b)] hop distances are presented. The MTTD of a failure increases as the number of cameras increases because the increased amount of video data increases the amount of computation that is performed in the cloud, which increases the time cost. Similar results are obtained for these cameras with the H4-hop distance in Fig. 2(b). The MTTD for the same number of failures is lower than that in Fig. 2(a), but the variation trend of MTTD is identical. In addition, all the MTTDs increase as more failures are detected from 450 cameras with the H4 and H8-hop distance [see Fig. 2(c)], although the MTTD is much smaller with H4 hops distance than that with H8-hop distance. The results demonstrate that the cloud-based approaches consume substantial computing resources for detecting failures in video data.

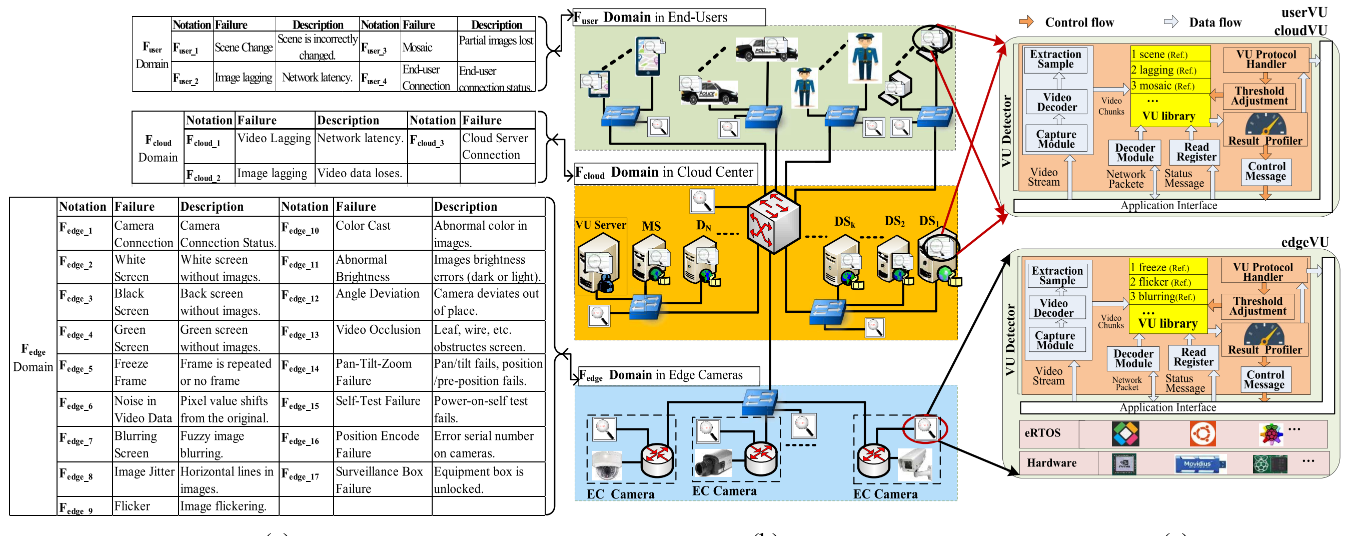
Summary: The results demonstrate that QoS or QoE problems commonly occur in video data. We observe an interesting phenomenon: video data satisfies QoS metrics, but the video content is useless due to an angle deviation failure or an obstacle failure in the camera. The useless video data is yet sent to the cloud as well. With high-resolution video data (e.g., 4K resolution) [14], this is highly wasteful of both the network bandwidth and storage usage in the cloud [15]. We conclude that the massive number of cameras and a long surveillance time render the detection of a failure in the large-scale video surveillance system scarcely possible as its scale is increased. The detection may be false due to human subjectivity, which poses new challenges in operations and maintenance teams. In addition, the cloud manages a large amount of video data with failures, which burdens the network and wastes storage usage in the cloud. To the best of our knowledge, there are few effective approaches to handling useless video data on the fly. A failure may occur in an edge camera, an end-user, the network, or a cloud server in a video surveillance system.

B. *Our Vision—Video Usefulness*

To address the above issues, we design a VU model for detecting a failure in a video surveillance system intelligently. The objective of the VU model is threefold: 1) analyzing video data from an edge camera and a data package from a network node (router or switch) in a fast online fashion; 2) detecting a failure efficiently via intelligent methods on the fly; and 3) sending prompt alarm messages of a failure to the operations and maintenance team and helping them recover it as fast as possible. Therefore, we employ edge computing [16] in this model for a large-scale video surveillance system. Edge computing has emerged along with the development of applications [17]–[19] in the realm of the Internet of Things (i.e., IoT) [20] and Internet of Everything (i.e., IoE) [21] in recent years.

To the best of our knowledge, we are the first to propose an edge computing-enabled failure detection framework for VU

[2]These failures include occlusion, offline, flicker, video lost, blurring, color cast, brightness, freeze, noise, jitter, scene change, and angle deviation.



	Notation	Failure	Description	Notation	Failure	Description
F_{user} Domain	F_{user_1}	Scene Change	Scene is incorrectly changed.	F_{user_3}	Mosaic	Partial images lost
	F_{user_2}	Image lagging	Network latency.	F_{user_4}	End-user Connection	End-user connection status.

	Notation	Failure	Description	Notation	Failure
F_{cloud} Domain	F_{cloud_1}	Video Lagging	Network latency.	F_{cloud_3}	Cloud Server Connection
	F_{cloud_2}	Image lagging	Video data loses.		

	Notation	Failure	Description	Notation	Failure	Description
F_{edge} Domain	F_{edge_1}	Camera Connection	Camera Connection Status.	F_{edge_10}	Color Cast	Abnormal color in images.
	F_{edge_2}	White Screen	White screen without images.	F_{edge_11}	Abnormal Brightness	Images brightness errors (dark or light).
	F_{edge_3}	Black Screen	Back screen without images.	F_{edge_12}	Angle Deviation	Camera deviates out of place.
	F_{edge_4}	Green Screen	Green screen without images.	F_{edge_13}	Video Occlusion	Leaf, wire, etc. obstructes screen.
	F_{edge_5}	Freeze Frame	Frame is repeated or no frame	F_{edge_14}	Pan-Tilt-Zoom Failure	Pan/tilt fails, position /pre-position fails.
	F_{edge_6}	Noise in Video Data	Pixel value shifts from the original.	F_{edge_15}	Self-Test Failure	Power-on-self test fails.
	F_{edge_7}	Blurring Screen	Fuzzy image blurring.	F_{edge_16}	Position Encode Failure	Error serial number on cameras.
	F_{edge_8}	Image Jitter	Horizontal lines in images.	F_{edge_17}	Surveillance Box Failure	Equipment box is unlocked.
	F_{edge_9}	Flicker	Image flickering.			

(a) (b) (c)

Fig. 3. Edge computing-enabled VU-based video surveillance system. Fig. 3(a) presents failures types observed in a large-scale surveillance system. Fig. 3(b) shows three failure domains (i.e., the F_{edge} domain in edge cameras, the F_{user} domain in end-users, and the F_{cloud} domain in the cloud center) in a video surveillance system, which are divided according to the failure types. Fig. 3(c) illustrates three VU (i.e., VU) frameworks, namely, edgeVU, userVU, and cloudVU for detecting failures of the video stream in three failure domains. VU, EC, DS, and MS denote VU, edge computing, a data server, and a meta-data server, respectively.

in video surveillance systems. This framework fully utilizes an edge computing paradigm to detect a failure and shorten the MTTD. It is an efficient method for saving bandwidth in the network and improving the utilization of video storage in the cloud. The VU model, which is deployed in edge cameras, end-users, network nodes, and cloud servers, can detect the state of any part of a video surveillance system and locate a failure on the fly. VU aims at realizing the following goals.

1) Early detecting a failure (or VU) in a video stream to shorten the MTTD for a failure.
2) Relieving the burden on the data transmission in the network and reducing the storage usage of useless video data.

In summary, the VU model efficiently preprocesses video streams according to its VU value. Video data that correspond to lower VU values need not be uploaded to the cloud, which reduces the burden in the network and improves the video storage usage in the cloud.

C. Contributions

Our main contributions in this article are as follows.

1) We explore a city-scale video surveillance system and identify three failure domains: a) the edge failure domain; b) the end-user failure domain; and c) the cloud failure domain. For each domain, we classify 17, 4, and 3 types of failures, respectively, one of which impacts the useful content of video streams [see Fig. 3(a)].
2) According to the failures in domains, we propose an edge computing-enabled VU model. VU empowers edge nodes of the network with computing capability for performing failure detection on the fly. VU filters useless video data in nodes to avoid network overload and improve the cloud storage usage. VU supports the usefulness of video for accurate video analytics in large-scale video surveillance systems [see Fig. 3(b)].
3) To detect failures in domains, we implement a fast-online edge computing enabled failure detection framework for studying VU in which several approaches of video/image processing are designed and migrated down to the edge node. This framework aims at detecting a failure in each node using edge computing [see Fig. 3(c)]. Then, early failures are accurately detected in real time to improve the MTTD.
4) We devise a failure-aware and dynamic adjustment (i.e., FADA) scheduling strategy for dynamically reordering the frequencies of failure detection approaches to tradeoff the consumption of computing resources and the performance of MTTD in the edge computing environment.

II. Failures and Failure Domains in Video Surveillance Systems

In this section, we explore failures that impact the usefulness of video data in a video surveillance system with approximately 4000 cameras at Anhui University. We classify the various failures into three groups, one of which is defined as a failure domain in the system [see Figs. 3(a) and (b)]. We propose a VU framework [see Fig. 3(c)] for detecting failures via several lightweight image processing approaches.

Failure Types [See Fig. 3(a)]: A video surveillance system is typically composed of cameras at the edge, video servers in the cloud, end-users, and the network. An edge camera acquires a video stream which is stored in a cloud server; then, video data is presented to an end-user. A failure in any component impacts on the usefulness of the video streams. To enclose failures in the system, we spent approximately eight months studying the video streams from approximately 4000 sampled cameras in the large-scale video surveillance system platform at Anhui University. Video data is characterized in terms of three qualities: 1) satisfactory QoS (or QoE); 2) poor QoS; and 3) useless content. For satisfactory QoS, video data is available for monitoring scenarios and further video data analytics. A satisfactory-QoS video may have useless content. For example, the content of a video from an illegal-angle camera cannot be used in video analytics. Video data with poor QoS cannot be used for further video analytics.

In an empirical case, a natural occlusion failure occurs, in which the camera screen is obstructed by bags, leaves, or wire. An end-user approach can detect a screen change failure in which a scene is incorrectly changed to another scene. In a video loss failure, video data are lost in a cloud server. We collected video data with 24 types of failures in three failure domains in a video surveillance system, see Fig. 3(a).

Failure Domains [See Fig. 3(b)]: We explored 24 failures which occurred in four components in a realistic video surveillance system, i.e., edge cameras, the cloud center, end-users, and the network. According to the location of each failure, we define a component with multiple failures as a *failure domain*. Therefore, we define three failure domains, namely, the F_{edge} domain, the F_{cloud} domain, and the F_{user} domain in a video surveillance system in Fig. 3(b). Failures on edge cameras constitute the F_{edge} domain. Failures in cloud servers (e.g., meta-data servers and data servers) and connected network nodes are grouped together to form the F_{cloud} domain. The end-user domain, which is composed of end-user nodes and failures, is referred to as the F_{user} domain. We explore 17, 4, and 3 types of failure in the F_{edge}, F_{user}, and F_{cloud} domains [see Fig. 3(a)], respectively. Any failure in one of these domains can impact the VU.

III. Video Usefulness Definition

As discussed above, the usefulness of a video depends on whether a failure occurs on each node in the three failure domains. We propose a VU model to study a failure and its impact on the amount of video data in a video surveillance system.

Our VU model detects the usefulness of the video data on any node in a video surveillance system. In the cloud, the amount of video data is largely determined by its usefulness. Thus, we define the VU in terms of the amount of video in a video surveillance system. The amount of useful video data is defined as follows.

Definition 1: The amount of useful video data is the size of the useful and failure-free portion of the entire video stream, which is transmitted to the cloud and stored in a data server over a period of time.

In a video surveillance system, let V represents the amount of video data which is captured from a camera in the time interval $[1, N_T]$, where the times are expressed in seconds. Video data that are uploaded to a storage server in the cloud cross many edge nodes. Let d_i be the amount of video data from the eth camera in the ith unit time. In the range $[1, N_T]$, the amount of video data is expressed as follows:

$$V_e = \sum_{i=1}^{i=N_T} (d_i). \tag{1}$$

The d_i video data will be transmitted along the path between the eth camera and the server in the cloud; therefore, it is assumed that there are M_{node} nodes in the path. VU is referred to as *VU value* (or *the degree of VU*) in this article. $U(j)$ indicates the usefulness value of the video data in the jth node (e.g., an edge camera and a router in the network) per unit time in a video surveillance system. Then, we have

$$U(j) \in [0, 1] \tag{2}$$

where $U(j) = 0$ denotes that the video data in the jth node is useless and cannot be uploaded to the cloud. Otherwise, $U(j) = 1$ denotes that the video data is useful and is uploaded to the cloud.

A value that is between *zero* and *one* is compared with a reference value that has been empirically set by the operations and maintenance teams. It must be noted that $U(j)$, which denotes the VU value of video data d_i on the jth node, is used to evaluate the VU. This metric is expressed as follows:

$$U(j) = \begin{cases} u, & \Delta v > 0 \\ 0, & \Delta v \leq 0 \end{cases} \tag{3}$$

where $j \in [1, M_{\text{node}}]$ and Δv refers to the usefulness boundary of the video data which depends on the cases. The parameter u, which corresponds to the value of VU and satisfies $u \in (0, 1]$, is determined based on the difference between the VU value and the user-defined reference, i.e., $\Delta v = |v_{\text{test}} - v_{\text{Ref}}|$. If $\Delta v > 0$, the video data from a node is useful for users. Therefore, a portion of the useful video data, the ratio of which is equal to u, is uploaded to the upper-level node from the lower-level node. Otherwise, if $\Delta v \leq 0$, the video data corresponds to a lower degree of VU than the reference value and the video data is useless and is directly discarded. The value of u, which is empirically set by the operations and maintenance teams, is qualified in each failure detection approach (see Section IV). In this article, the value of u is set to one in each detection approach in our experiments.

Through all nodes in the video surveillance system, the real amount of video data in the video data d_i from the eth camera is expressed as follows:

$$D_i = d_i \times \prod_{j=1}^{j=M_{\text{node}}} U(j) \tag{4}$$

where M_{node} denotes the number of nodes in the path of the video data between a camera and a cloud server. D_i denotes the amount of video data in the cloud.

Then, the amount of useful video data that are captured from the eth camera and stored in the cloud is represented as follows:

$$V_e^u = \sum_{i=1}^{i=N_T} (D_i) = \sum_{i=1}^{i=N_T} \left(d_i \times \prod_{j=1}^{j=M_{\text{node}}} U(j) \right) \tag{5}$$

where V_e^u is the amount of useful video data that is stored in the cloud from the eth camera in the time range $[1, N_T]$.

Let S be the number of cameras in the video surveillance system. The amount of video data V_t^u is given as

$$V_t^u = \sum_{e=1}^{e=S} \left(V_e^u\right) = \sum_{e=1}^{e=S} \left(\sum_{i=1}^{i=N_T} \left(d_i \times \prod_{j=1}^{j=M_{\text{node}}} U(j) \right) \right) \tag{6}$$

where V_t^u is the amount of useful video data stored in the cloud from the $\boldsymbol{S}$ number of cameras in the video surveillance system during $[1, N_T]$ seconds.

In this article, the two methods are designed for handling video uselessness in a video surveillance system. First, edge devices directly terminate video data transmission to the cloud according to the VU value. Second, the VU value is sent to end-users for a further decision. The absolute value of v, which is denoted as $|v|$, represents a boundary value of VU for the video data. It divides the VU into various degrees. The value of $|v|$ is dependent on failure detection approaches.

IV. Failure Detection Framework

In a video surveillance system, video streams with failures can cause useless content, which burdens the network bandwidth and wastes storage usage in the cloud. Currently, a video failure is detected by examining video data in the cloud, which requires substantial time and computing resources.

Our proposed failure detection framework [see Fig. 3(c)] employs an edge computing paradigm in edge nodes to analyze the VU in a video surveillance system. Prior work [16] defined the "edge" as any computing and network resources along the path between data sources and cloud data centers. The design of a failure detection framework is determined by the available computing resources for detection approaches in three domains, such as computing resources that are equipped inside the device, newly incorporated computing units, and local (or no-additional) computing resources in the cloud and end-user. These resources are referred to as edge computing units and provide computing capabilities for edge nodes, e.g., edge cameras, network nodes, end-users, and cloud servers. Therefore, this framework preprocesses video streams or network packets to early detect a failure in each edge node along the path between a camera and a cloud server.

For example, a camera cannot conduct failure detection because of its limited computational capability. Thus, we then configure an edge computing-based camera (i.e., EC camera), which includes an edge computing unit and a camera that is connected to a router, as shown in Fig. 3(b). A software stack is deployed on an edge computing unit which preprocesses the video streams before it is uploaded to the cloud.

In this article, we designed three types of frameworks to detect failures for VU by using different computing resources above, i.e., *edgeVU*, *cloudVU*, and *userVU* frameworks. An additional computing unit connected to edge nodes by a router enriches the computation capability of the computing resources-limited devices, e.g., cameras, router, etc. An edge computing resources-based framework is called *edgeVU* [see Fig. 3(c)] in this article. The *cloudVU* and *userVU* frameworks use local computing resources to perform failure detection in the cloud and end-user, respectively. The former, which is deployed in a cloud server and use its computing resources, is called *cloudVU*. The *userVU* framework is designed in an end-user device. In addition, an end-user device also uses *edgeVU* to detect failures, e.g., scene change, video lagging, etc. Details of *edgeVU*, *cloudVU*, and *userVU* frameworks are listed as below.

A. EdgeVU Framework

The *edgeVU framework* depends on both additional hardware and software, as illustrated in Fig. 3(b), because an edge camera is a computing-resource-limited device. Many embedded hardware platforms, which provide computing power, are employed as edge computing units in this framework, e.g., Raspberry Pi V3 [22], Intel Movidius [23], and NVIDIA Jetson TX2 [24]. Besides, we employ a lightweight real-time operating system as an edge computing-oriented operating system (i.e., eRTOS). A VU detector, which is a user-level module, transfers messages to/from eRTOS through application interfaces (APIs). A *VU Library* module includes several detection approaches. In the edge camera domain, the edgeVU captures video data from a camera using the *Capturing Module* and executes failure detection approaches to determine the VU in various ways. For example, network packets are detected by the *Decoder Module*. The *Threshold Adjustment* module sets parameters for a detection approach at different resolutions. The detection results (e.g., latency and accuracy) are recorded by a *Result Profiler* module. The *Control Message* module analyzes the result and makes a decision on whether to adjust the camera, router, or others nodes. Meanwhile, the results are packed as a message by the *VU Protocol Handler* module and sent to a VU server (i.e., VU server) in the cloud. A VU server, which receives VU values from the *Result Profiler Module* and monitors the status of the framework on each edge node, stores log files of VU values in the cloud. An end-user accesses the VU server and searches for the VU value of the data in a node. In addition, the VU server pushes new versions of failure detection approaches onto an edge node in the online mode according to the requirement of failure detection. Developers design customized software in the function module.

B. CloudVU and UserVU Frameworks

The *cloudVU framework* is deployed in the cloud domain (e.g., data and meta-data servers) and the userVU framework is installed in the end-user domain (e.g., personal computers, police vehicles, and mobile phones). The two frameworks have the same structure and detect failure video data by using local computing resources in a cloud server (e.g., data center) or a mobile phone, which is sufficient for satisfying the requirements of the detection approaches in the VU library.

The approaches in the VU library are reconfigured for detecting a failure. For example, in the edge domain, the VU is highly important; thus, the detection approaches for video data are installed in the edgeVU framework. Video data after the edge domain is considered to be normal. However, the failure may occur when data packet transmits among the network, data server, and a meta-data server. Therefore, the detection approaches should be reconfigured in the cloud or the end-user domain. To record the VU values of the three domains, all detection results from the *Result Profiler Module* are sent to a VU server in the cloud.

In summary, an edge computing paradigm-enabled VU framework aims at detecting a failure on the fly by using video data or network packets on edge nodes in the edge computing model (not cloud computing model) in a video surveillance system. Video failure detection framework, which is based on the edge computing network, is considered a killer application for edge computing.

V. Failure Detection Approaches and FADA Scheduling

We built upon several lightweight approaches for detecting video failures in three domains, which differ in terms of the associated failure detection approaches. Edge computing-enabled analysis approaches are applied to video streams. The results are sent to a VU server in the cloud and end-users are notified. Detection modules handle useless video and decide whether to send the video to the cloud but a portion of the decisions will be made by users. First, we present several lightweight detection approaches in three domains in Sections V-A1–V-A3. Then, we proposed a scheduler, namely, FADA for determining the optimal detection interval for failure detection approaches and realize a high-performance failure detection framework.

A. Failure Detection Approaches in Domains

Several failure detection approaches are designed and deployed in each domain.

1) Failure Detection in the F_{edge} Domain:

a) Solid color screen and freeze frame detection: Solid color screen failures include $F_{\text{edge\_2}}$ (white screen), $F_{\text{edge\_3}}$ (black screen), and $F_{\text{edge\_4}}$ (green screen), which are presented in Fig. 3(a). The failure is manifested as a solid color screen because video transmission stops for a period of time. These approaches are utilized in edge computing-enabled cameras due to their lower cost in terms of computing resources. A detection approach searches for solid color images in video streams and calculates the duration. $F_{\text{edge\_5}}$ (freeze frame) occurs when the last image frame is repeated or no frame arrives for a long time.

An RGB image is converted to a grayscale image, and the gray values of all the pixels are considered. The differences in the gray values among the pixels correspond to the variability degree of the image. If the screen is solid, the differences in the gray values among the pixels are very small. A threshold gray variance value [25] is set for pixels in an image with a solid color failure. This failure is detected by comparing the threshold value with the variance for pixels in the image. For $F_{\text{edge\_5}}$, the variance of the interframe difference is calculated for comparison with a reference. The N subsequent frames are detected to confirm this failure. The time stamp in a video stream with a freeze failure is unchanged, whereas it is variable in a static scene for a normal image.

b) Noise detection: A noise failure ($F_{\text{edge\_6}}$) is a random variation of the brightness or color information in an image. Gaussian noise [26] has a probability density function that is equal to a normal distribution. Salt-and-pepper noise [27], which is defined as impulsive noise, is characterized by dark pixels in bright regions or bright pixels in dark regions of a video. The absolute values of the gray differences between a pixel and its neighboring pixels in eight directions are employed to determine whether the test video contains noise. When the minimum value exceeds a threshold, this pixel is a noise point. A Gaussian-type membership function is used. The pixel is a noise pixel if the corresponding value of the membership function [28] is smaller than a reference value. Then, the noise density is estimated from the number of noise points. The test video contains noise when the value exceeds a reference value.

c) Blurring failure (F_{edge_7}): A blurred image has reduced sharpness and deformations of edges. It resembles a video that is viewed through a translucent screen. The test image is divided into blocks for edge detection [29]. The image sharpness (S for example) is calculated based on the fuzzy probabilities of these small blocks. This failure is detected by comparing the value of S with a reference value.

d) Angle deviation and scene change detection: For an angle deviation failure ($F_{\text{edge\_12}}$), both the test and reference images at the edge are used as input parameters of speeded-up robust features function (SURF) [30]. Feature points in the two images are calculated. The two sets of feature points are configured to be parameters of the *BruteForceMatcher* function and the set of matching-point pairs between the two images; for example, $p_i(x_i, y_i)$ and $p_j(x_j, y_j)$ are a pair of matching points. The offset of and Euclidean distance between the two matching points can be calculated as $\Delta p_i = |p_i(x_i) - p_j(x_j)|$ and $\Delta p_j = |p_i(y_i) - p_j(y_j)|$, respectively. We then calculate $\text{dist}(p_i, p_j)$. A pair of matching points in the angle deviation is counted when the three values all exceed their corresponding reference values. When the number of angle-deviation matching points exceed a reference value, the test image has an angle deviation failure.

In addition, a scene change failure [see $F_{\text{user\_1}}$ in Fig. 3(a)] occurs when a normal scene incorrectly changes. This failure is detected via a similar method to the angle-deviation failure detection method using a different threshold.

e) Flicker screen detection: Frames f_{i-1}, f_i, and f_{i+1} are extracted from the video stream to calculate two variances of the interframe difference between two adjacent frames [25]. A flicker failure ($F_{\text{edge\_9}}$) is detected by comparing the difference between the two variances with the reference value.

f) Color cast detection: An image that has been extracted from the video is converted into the *Lab* color space [31]. For an image with a color cast failure, the mean values in dimension a and b deviates from the original one and the variance is small. This feature is helpful for obtaining the color cast factor, which is used to determine whether a failure has occurred.

g) Abnormal brightness detection: A test RGB image is converted into the hue, saturation, intensity (HSI) color space [32]. The number of pixels in the image for which exceeds a threshold is calculated. If the ratio of the number of these pixels to the total number of pixels in the image exceeds a reference value, an abnormal brightness failure ($F_{\text{edge\_11}}$) is detected. Then, the HSI method, which reflects the saturation and brightness in an image, detects the failure $F_{\text{edge\_11}}$.

h) Image jitter detection: A jitter failure ($F_{\text{edge\_8}}$) results from the horizontal lines of images being randomly displaced due to the corruption of synchronization signals or electromagnetic interference in transmission. The same method as for angle deviation detection [see (4) in Section V-A1] is employed for the detection of this failure.

i) Video occlusion detection: For an occlusion failure ($F_{\text{edge\_13}}$), a background image that corresponds to a reference value is extracted from the video in the first 2 s. An image from the subsequent 2 s of video is extracted and its interframe difference with the background image is calculated to obtain a foreground image [33]. After the image processing corrosion expansion operation, the area of the new image is defined as *img1Area*. By region growing [34], the area of the block of maximum area in the foreground image (*maxArea* in short) is calculated according to *img1Area*. Let SCALE be the ratio of *maxArea* to *img1Area*. The variance of the region that corresponds to *maxArea* and the same region of the current image is denoted as VAR. If SCALE $>$ T1 and VAR $>$ T2 (T1 and T2 are reference values), there may be an occlusion failure in the image. The single image that is extracted from the subsequent 2 s of video data is processed via the above steps. The video has an occlusion failure if occlusion failures are detected in consecutive 24 frames.

j) API-based failure detection: Failures $F_{\text{edge\_14}} \sim F_{\text{edge\_17}}$ are detected by the APIs in cameras that are provided by the manufacturers.

For $F_{\text{edge\_14}}$, the pan/tilt module in a pan-tilt-zoom (PZT) camera fails to work. $F_{\text{edge\_15}}$ is an initial failure in the power-on-self test. $F_{\text{edge\_16}}$ exists in the position encoded for a camera. The position that is encoded number, which is displayed on an LED screen, represents an edge camera and provides convenience in retrieving its video for users. The digital number is recorded by the detection system for comparison with the correct number in the local storage device. In a surveillance equipment box failure (i.e., $F_{\text{edge\_17}}$), the equipment box is unlocked. The detection framework reads the register status in a camera using its APIs for PTZ, self-testing, the number in LED, and the equipment box.

k) Edge camera connection detection: For $F_{\text{edge\_1}}$, two types of failure occur in the network, i.e., disconnection and problematic connection. Using the feedback of a TCP/IP data packet, the detection approaches vary according to the failure in this case.

For *a disconnection failure*, the detection module, which is deployed in the cloud or the end-user domain, sends test data packets to a targeted edge camera and waits for the response. If the detection framework determines that the allowed response time has been exhausted, the network disconnection is reported to an end-user. If response packets arrive, the detection approach identifies the serial numbers of the data packets and compares them with the configured serial numbers of the request packets. When the serial number of a response packet is zero or is the same as that of a request data packet, the network is connected but has a problem. This case is classified as a *problematic connection failure*. Finally, the prompt for $F_{\text{edge\_1}}$ is sent to the end-users or stored in the VU server.

2) Failure Detection in the F_{user} Domain: There are four types of failure in this domain. Computer vision techniques are also used in the end-user domain because end-users browse video clips and images.

a) Video lag detection: The network latency causes a video lag failure (see $F_{\text{user\_2}}$), which is detected according to the time interval between a request data packet being sent to an edge camera (or a cloud server) and the response packet being received. In addition, the packet loss ratio for fast online video is calculated from the average number of response packets and the latency.

b) Mosaic failure detection: A mosaic failure ($F_{\text{user\_3}}$) occurs due to the low quality of the transmission data in the network. Images can be dropped in a low-quality network, which is defined as mosaic failure. To detect this failure, we employ a mosaic-square reference model for performing model matching after *sobel edge detection* for a test image [35]. In the image, the number of a right angle is calculated for comparison with a reference value.

c) End-user connection detection: The detection method for failure $F_{\text{user\_1}}$ is similar to that for $F_{\text{edge\_1}}$, which detects disconnections and problematic connections using the feedback of TCP/IP data packets. A detection approach in the VU library that is deployed in the edge or the cloud domain sends a test data packet to an end-user and waits for a response packet. The status of the end-user offline ($F_{\text{user\_1}}$) notifies the end-users in other ways and is stored in the VU server, please refer to the section on $F_{\text{edge\_1}}$ detection.

3) Failure Detection in the F_{cloud} Domain: It is difficult for humans to manage large-scale video data and timely detect failures online in the cloud. If a failure occurs in a cloud server, video service is impaired when the historical video is applied. Three detection approaches are designed for locating failures in the cloud.

a) Cloud server connection detection: $F_{\text{cloud\_1}}$ [see Fig. 3(a)] occurs in two types of network connection: 1) from an edge camera to a cloud server and 2) from an end-user node to a cloud server. The detection system is configured at an edge or an end-user domain, respectively. For this problem, disconnection and problematic connection are detected by using the feedback from TCP/IP data packets, similar to $F_{\text{edge\_1}}$ (or $F_{\text{user\_1}}$).

b) Video lag and loss detection: A video lag failure is discovered when an end-user looks up a video, which is unacceptable in time-sensitive criminal cases. Detection approaches in the VU library check the network latency in the cloud. The latency refers to the duration from a request packet being sent to an edge camera to the arrival of the response data packet. The packet loss ratio is used to measure the number of data

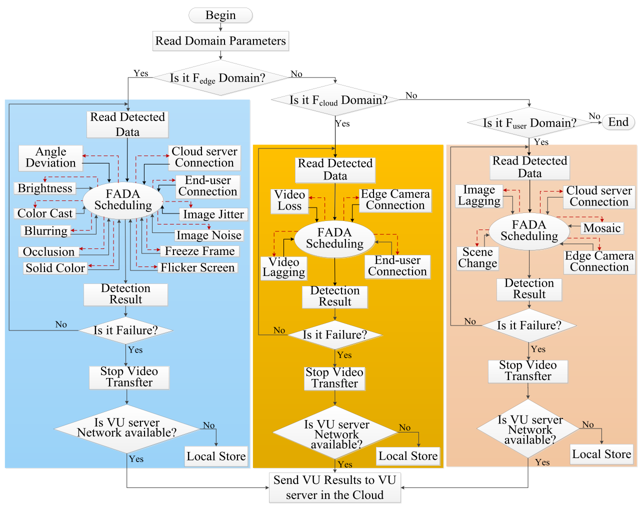


Fig. 4. Workflow of failure detection approaches using FADA in edge computing-enabled VU systems. The *black solid line* and *the red dashed line* represent the data stream and the control stream, respectively, in each failure domain.

packets for video data. When the value is lower than a reference value, the usefulness of the video is low. $F_{\text{cloud\_3}}$ (video loss) is caused by a severe packet loss.

B. FADA Scheduling Scheme

According to the failure detection approaches that are described above, we present three types of workflows of failure detection approaches in the edge, cloud, and end-user failure domains, as shown in Fig. 4. In each workflow, we propose a scheduler, namely, FADA. FADA schedules failure detection approaches according to the frequency of failure occurrence. Then, FADA sends the results to a VU server in the cloud if the network is available. Otherwise, the results are temporarily stored in the local device. In addition, FADA adjusts the frequencies of failure detection approaches and configure the optimal detection time interval for failures. The *detection interval* for a failure detection approach refers to the time interval between the detection of two consecutive failures of the same type.

In the edge model, the high frequencies of the failure detection approaches improve the MTTD. However, the edge device still cannot afford the cost of energy consumption due to its limited resources. Thus, it is important to design a scheduling scheme that dynamically adjusts the frequencies of failure detection approaches to balance the MTTD and the frequency.

In FADA, the frequency of failure detections depends strongly on the time interval between applications of the same detection procedure. Initially, the time interval between two failure detections using the same procedure predefined. It is supposed that the first detection procedure for the jth failure starts at $t_{(j,i)}$ and the same detection procedure is performed again at $t_{(j,i+1)}$. Let us suppose that the current time interval between the two applications of the same failure detection approach is T_D. If we have

$$\left(t_{(j,i+1)} - t_{(j,i)}\right) < T_D \tag{7}$$

Algorithm 1 FADA Scheduling

```
1: procedure FREQUENCY
2:   $T_D$ ←the time interval for each failure detection in the edge, cloud and user domain.
3:   $sum$ ← the times of consecutive failure
4:   $sum1$ ← the times of consecutive normal
5:   $tag$ ← the flag of the approach
6:   $Sa$ ←the sequence of the approaches $(F_1, F_2, F_3, \ldots, F_{max})$
7:   Subscript $max$: the number of detection approaches in an domain.
8:   while true do
9:     Runing Sa
10:    $tag$ ←the flag of $F_1$
11:    if $F_j$ prompt camera failure then
12:      OutPut("Video Surveillance failure")
13:      Waiting for repair
14:      $sum1$ ←0
15:      if tag == the flag of $F_j$ then
16:        sum++
17:      else
18:        sum←0
19:        tag←the flag of $F_j$
20:      end if
21:      if sum > 3 then
22:        $T_D$ ← $T_D/2$
23:        set the $F_j$ as the first approach
24:      end if
25:    else
26:      $sum$ ←0
27:      sum1++
28:      if sum1 > 3 then
29:        $T_D$ ← $T_D \times 2$
30:      end if
31:    end if
32:  end while
33: end procedure
```

the occurrence frequency of this failure is high. When the same failure is consecutively detected N_{Ref1} times (e.g., $N_{\text{Ref1}} = 3$ in FADA), FADA increases the frequency of the failure detection and adjusts the detection approach for the jth failure as the first detection procedure among all failure detection approaches, namely, the time interval between two failure detections is shortened. We then redefine the time interval as follows:

$$T_D = T_D/2. \tag{8}$$

Hence, the frequency of the jth failure detection is doubled. Meanwhile, we readjust the frequencies of other failure detections. Other failures may occur in the same case; therefore, the adjustment of the frequency of this failure detection improves the probability of detecting failures in real time.

Otherwise, no failure is detected in video streams; and this case is repeated N_{Ref2} times (e.g., three times in this article). Then, the time interval between the two failure detections using the same procedure increases and the frequency of failure detection decreases, e.g., $T_D = 2 \times T_D$. If the following three detection procedures do not detect failures; then, the time interval between two failure detections is set to $(2 \times T_D)$ until a failure is detected. Details are shown in Algorithm 1. In addition, the interval time between two application of two failure detection approaches in a round is also increased to improve the MTTD.

In summary, the FADA scheduling scheme can realize two goals: 1) finding a failure fast online in a tradeoff frequency in the edge model with the limited resources and 2) improving the MTTD and the frequency for failure detection via the FADA scheduling strategy. We evaluate the performance of FADA in terms of the MTTD, the frequencies at which the detection approaches are applied, and the energy consumption. Details of the evaluation are presented in Section VI-G.

VI. Experiment and Evaluation

The benefits of the edge computing-based VU model in a video surveillance system are examined to answer the following questions.

Q1: Why can the VU model accurately evaluate the VU? How does the VU model work?

Q2: Why can the VU model early and efficiently detect a failure in a video surveillance system and shorten the MTTD? How dose the VU model work?

A. Accuracy of the VU Model

In the dataset, the number of true- (or false-) positive and negative test data items (images or video clips) are counted. The binary classification tradeoff [36] is used to evaluate the accuracies of detection approaches. We classify the detection results into two categories, i.e., normal and failure. The VU model detects categories approximately and the classification results are listed as follows.

True Positive (TP): A failure video data item is correctly detected as a failure $\surd$.

False Negative (FN): A failure video data item is mistakenly detected as a normal $\times$.

True Negative (TN): A normal video data item is correctly detected as a normal $\surd$.

False Positive (FP): A normal video data item is mistakenly detected as a failure $\times$.

It is supposed that the dataset contains $\boldsymbol{N}$ (i.e., $N = (I+O)$) data items. I and O represent the number of true failure and normal data items, respectively. Four types of accuracy ratios (i.e., *efficiency, precision, recall*, and *specificity*) are employed to evaluate the accuracy degree of the detection approaches in the VU model. The *efficiency* ($\boldsymbol{E}$) is the difference between the test value and the true one for a detection approach. This metric measures how close a test value is to the true value. $\boldsymbol{E}$ is defined as $E = ((\mathrm{TP} + \mathrm{TN})/N) \times 100\%$. The *precision* ($\boldsymbol{P}$) is the repeatability of successive measurements under the same conditions. It is the positive predictive value for evaluating how the test values are close to each other and; is defined as $P = (\mathrm{TP}/(\mathrm{TP} + \mathrm{FP})) \times 100\%$. The *recall* (true acceptance rate, $\mathbf{R}$), which is the fraction of true failure video data items in the dataset that are detected as failure items, is expressed as $R = (\mathrm{TP}/I) \times 100\%$. The *specificity* (true rejection rate, $\mathbf{S}$) is the fraction of true normal video data items in the dataset being detected as normal items. $\boldsymbol{S}$ is defined as $S = (\mathrm{TN}/O) \times 100\%$.

In this article, we select 100-failure and 60-normal data items. We have $I = (\mathrm{TP} + \mathrm{FN}) = 100$ and $O = (\mathrm{TN} + \mathrm{FP}) = 60$. Without loss of generality, we use two parameters, i.e., *TP* and *TN*, in the experiments.

B. Experimental Setup

1) Evaluation Dataset: In the experiment, the dataset is collected from a realistic video surveillance system that has approximately 4433 IP cameras at Anhui University. In addition to 4232 cameras online cameras, there are 201 cameras that are unable to run, which is attributed to: 1) failure of a charge-couple device (CCD) in a camera; 2) CMOS imaging sensor damage; and 3) a camera being past its useful life. In the video data from 4232 online cameras, failures occur, such as occlusion and flicker. Because each IP camera is installed in a distinct location, there are 4232 types of scenes in video streams. We spent approximately six months collecting video clips and images with various failures from various scenes in the video surveillance system. According to realistic failures in the video streams, we designed a dataset that includes two types of video data, i.e., video clips and images. Each video clip or image, which is called a test item, can be classified as normal or a failure. There are ten types of failures (see Table I) that can occur in the dataset. These items are realistic candidates and hand-segmented video clips or images in various scenes. A failure is detected by preprocessing video clips, such as *angle, freeze frame, jitter, occlusion*, or *flicker*. Meanwhile, an image is preprocessed to analyze whether a *blur, brightness, color cast, solid color,* or noise failure occurs. For a failure test, we select 160 test candidates (items) including 100-normal and 60-failure clips or images, each of which contains a different scene. The number of failure data items in the entire set of 160 test candidates is configured as the distribution of failure items (e.g., 60 data items for angle deviation failure) in Table I. A data item in a scene has four resolutions, i.e., 1920×1080, 1280×720, 720×576, and 352×288. The date items in the dataset have a wide variety of sizes, such as approximately 1.7 MB to 9.6 MB for video clips and 2.4 KB to 1.5 MB for images.

Our goals of designing the dataset are twofold: 1) classifying video streams failures in a large-scale video surveillance and 2) evaluating failure detection approaches (or algorithms) in our proposed VU detection framework in terms of accuracy and MTTD on the dataset. In addition, we hope that this dataset will be helpful for researchers in evaluating the reliability of video surveillance systems.

2) Experimental Environment: According to failures, we classify the data items in the dataset for VU detection into ten groups. We conducted two types of experiments, i.e.: 1) basic evaluation and 2) scalability evaluation. To evaluate the basic accuracy of VU, we use 160 test video data items, including video clips or images to evaluate this metric in Section VI-C. We employ 100, 300, 500, and 700 cameras in the test for MTTD. In the scalability evaluation (Section VI-D), we extend the test dataset to approximately 4000 data items. We compare the VU platform with a real-world commercial failure detection platform that is provided by Uniview Corporation.[3] Six typical failure detection approaches (see **B**, **C**, **D**, **F**, **H**, and **I** in Table I) are considered in this test. This failure detection system is deployed in a real scenario at Anhui University.

Currently, no camera has an edge-computing unit; therefore, we build an edge computing-enabled camera platform using Raspberry Pi V3 (i.e., RPi V3). RPi V3 is equipped with a 1.2 GHz, 64-bit quad-core ARMv8 CPU. The test dataset is stored in a device on the RPi V3 testbed. We employ lightweight OpenCV 4.0 to implement the detection

[3]Uniview is an IP video surveillance cameras company in China.

TABLE I
FAILURES AND DISTRIBUTION IN THE EVALUATION

Symbol	A	B	C	D	E	F	G	H	I	J
Definition	Angle Deviation	Blurring	Brightness	Color Cast	Freeze Frame	Solid Color	Image Jitter	Occlusion	Noise	Flicker
#Failure	60	60	60	60	60	60	60	60	60	60
#Normal	100	100	100	100	100	100	100	100	100	100

Ten types of failures are considered in this study. #Failure and #Normal represent the numbers of failure and normal data items, respectively, for a failure-detection-approach test at each of four resolutions (*e.g.,* 1920×1080).

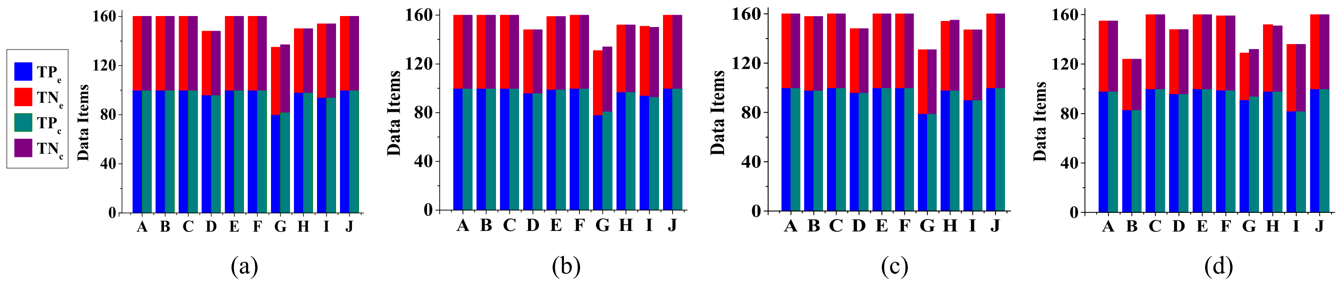


Fig. 5. Accuracy of failure detection approaches (see Table I) with four resolutions. TP_e (TP_c) and TN_e (TN_c) denote that the *TP* and *TN* results for ten failure detection approaches in the VU library in the two models. Subscripts *e* and *c* represent the edge- and cloud-based models, respectively. (a) 1920 × 1080. (b) 1280 × 720. (c) 720 × 576. (d) 352 × 288.

approaches. The RPi testbed is wired to the cloud. This cloud platform is emulated by six processors and 8 GB of memory in a Dell PowerEdge R730 server.

We conduct extensive experiments for evaluating the performance of the edge computing-based VU model (*edge model* in short) and comparing it with that of the cloud computing-based model (i.e., *cloud model*) in terms of accuracy and MTTD. MTTD is the sum of the transfer time and the execution time which is referred to as *the latency*. The edge computing-enabled component captures video data from a camera and detects the failure. The transfer time is relatively small in the edge model.

C. Performance Evaluation

In the edge domain, $F_{\mathrm{edge\_14}}$ (PZT failure), $F_{\mathrm{edge\_15}}$ (self-test failure), $F_{\mathrm{edge\_16}}$ (position encoding failure), and $F_{\mathrm{edge\_17}}$ (surveillance box failure) are detected via API-based methods. Failures of the offline (for an edge, an end-user, and a cloud server) and the image (or video) lags are measured via the network detection. The detection approach for scene changes is the same as that for angle deviations. Thus, we evaluate the remaining detection approaches in terms of accuracy and MTTD in the VU model. White screen, black screen, and green screen failures have the same features and turn into a single-model screen without images; they are referred to as solid color screen failures. Then failures (see Table I) are detected by the VU model.

In the edge model, these failure detection approaches are applied in parallel; then the VU values are sent to the VU server and the operations and maintenance teams. The MTTD of a detection approach is significantly smaller than that in the cloud model. Video with a low degree of VU need not be uploaded to the cloud, thereby reducing the use of storage in the cloud.

1) Accuracy of the VU Model Evaluation: According to parameters in Section VI-A, we conduct experiments to evaluate the accuracy of the VU model in terms of accuracy of the failure detection approaches. According to Figs. 5 and 6, the metrics for validating the accuracy of a failure detection approach include TP and TN. The *efficiency*, *precision*, *recall*, and *specificity* (*E, P, R,* and *S* for short) are used to evaluate the degree of accuracy.

a) Basic accuracy: Two basic metrics, namely, TP and TN, are used to evaluate the accuracy of failure detection approaches. In the dataset, there are 160 data items (images or video clips) with 100-failure and 60-normal data items for each failure-detection approach test at each resolution. Resolutions of 1920 × 1080, 1280 × 720, 720 × 576, and 352 × 288 for data items are used in the experiments. A video clip or an image, the size of which varies with the resolutions, consumes various amounts of computing resources and yields various degrees of accuracy. Fig. 5 presents the accuracy metrics (TP and TN in the edge and cloud models. For example, TP_e and TP_c represent the number of *TP* data items in the edge and cloud models, respectively.

The *TP* and *TN* values for the edge and cloud models are higher than those in Fig. 5(a)–(d). The results demonstrate high accuracies of the detection approaches in the VU model. For a failure detection approach, a larger sum of *TP* and *TN* corresponds to a more accurate detection approach. We have $(\mathrm{TP} + \mathrm{FN}) = 100$ and $(\mathrm{TN} + \mathrm{FP}) = 60$. The real number of failure and normal data items are 100 and 60, respectively. The closer that the sum of *TP* and *TN* is to 160, the more accurate the detection approach is. Considering blurring failure detection as an example [see Fig. 5(a)], the TP_e (or TP_c) is 100 and TN is smaller than 60; hence, all failure data items are correctly detected. Similar results are presented in Fig. 5(b)–(d).

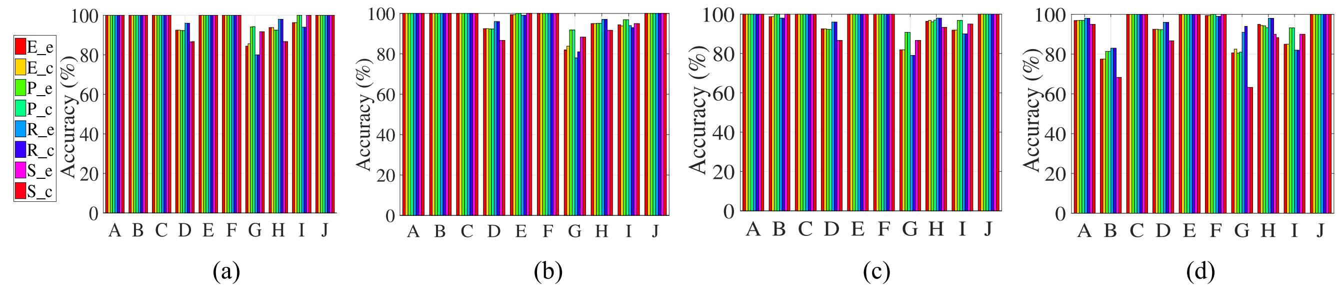


Fig. 6. Accuracy degree of failure detections at four resolutions. E_e (E_c), P_e (P_c), R_e (R_c), and S_e (S_c) denote the efficiency, precision, recall, and specificity, respectively, of a failure detection approach in the edge (cloud) model. (a) 1920 × 1080. (b) 1280 × 720. (c) 720 × 576. (d) 352 × 288.

According to Fig. 5(a)–(d), *TP* and *TN* exhibit similar phenomena at the four resolutions, e.g., abnormal brightness, color cast, freeze frame, solid color, video occlusion, and flicker screen. These values for failure detection approaches regarding blurring screen and noise become worsen as the resolution of the images or video clips decreases. The same detection approach is used for image jitter and angle deviation, which is based on feature extraction (see Section IV). The two failure detections results have the same trends in terms of *TP* and *TN*. Higher accuracy is realized at a higher resolution. In addition, the values of *TP* and *TN* are 100 and 60, respectively, for mosaic failure detection in the edge and cloud models.

b) Degree of accuracy evaluation: In this test, we analyze the accuracy degrees of detection approaches in the VU model in terms of the efficiency (**E**), precision (**P**), recall (**R**), and *specificity* (***S***) in edge and cloud models. Subscripts *e* and *c* of E_e and E_c denote the edge model and the cloud model, respectively. The ideal value for each metric is 100%.

At each resolution, the values of *E*, *P*, *R*, and *S* for abnormal brightness, freeze frame, solid color failure, and screen flick are mostly no less than 99.0% in Fig. 6. The results demonstrate a high degree of accuracy for these failure detection approaches.

With a low resolution, the values of detection for angle deviation, blurring, image jitter, and noise become smaller than those for the high resolution. For example, *E*, *P*, *R*, and *S* of the image jitter detection are much worse, i.e., 80.6%, 80.5%, 91.0%, and 63.3%, respectively, for the edge model and 82.5%, 81.0%, 94.0%, and 63.3% for the cloud model at 352 × 288 resolution. At 1920 × 1080 resolution, the value of *E*, *P*, *R*, and *S* is 84.4%, 94.1%, 80.0%, and 91.7% for edge model and 85.6%, 94.3%, 80.0%, and 91.7% for cloud model, respectively.

In addition, each value of *E*, *P*, *R*, or *S* for a mosaic failure-based detection approach is 100% at all four resolutions in both models.

2) MTTD of VU Model Evaluation: This test aims at validating the early detection performance of the VU model. The cloud model-based detection approaches are conducted on video data from cameras to the cloud. However, VU-based detection approaches are performed in an edge node without video transmission. Thus, the MTTD for the edge computing-enabled failure detection approaches of the VU model is smaller than that in the cloud model, see Fig. 7.

We conducted two tests to evaluate the MTTDs of failure detection approaches. *Type-1:* The failure detection approaches are all performed in the edge model; and *Type-2:* The failure detection approaches are all performed after uploading the test data items in the cloud model.

MTTD depends strongly on the sizes of the data items and the video resolution. A large-size or high-resolution data item requires more time for failure detection in video streams. In this article, we use video clip- and image-based data items to evaluate MTTD because a video clip is larger than an image. As listed in Table I, five failure detection approaches are based on video clips and five failure-detection approaches use images. Without loss of generality, we use the image-based brightness and the video clip-based flicker detection approaches to evaluate the MTTDs of VU at the four-type resolutions. The MTTD is the sum of the transfer time and the execution time.

Fig. 7 shows the MTTD for brightness failure (**C**) and flicker (**J**) in Table I. The number of data items is configured as 100, 300, 500, and 700, which emulates the images or video clips from cameras. In the edge model, the transfer time is small because the test data items are locally handled without being uploaded to the cloud. The detection approaches are conducted in EC cameras in parallel. For failure ***C*** detection, the transfer time is 0.09 s, 0.03 s, 0.02 s, and 0.01 s at resolutions of 1920 × 1080, 1280 × 720, 720 × 576, and 352 × 288, respectively. This metric increases to 9.46 s, 2.94 s, 2.13 s, and 0.80 s for the video clip-based detection approach for failure **J**.

The transmission time and execution time depend strongly on the sizes of test video clips and the images in the cloud-model. The transfer time accounts for the most of the total time in this model. For a 1280 × 720-resolution test, the transfer time for the image-based brightness detection is 146.1 s, 440.0 s, 712.5 s, and 991.5 s for 100, 300, 500, and 700 data items in the cloud model. The video clip-based flicker detection exhibits 2229.3 s, 6688.0 s, 11146.7 s, and 5605.3 s latency in data transmission for 100, 300, 500, and 700 data items, respectively. In addition to the sizes of the data items, the execution time also depends on the number of processors. The execution time for the brightness detection is 3.0 s, 1.6 s, and 1.2 s using two, four, and six processors, respectively. But the video clip-based detection costs 136.3 s, 77.5 s, and 54.2 s using two, four, and six processors, respectively. However, the edge model spends about 0.4 s on conducting failure detection in distributed cameras in parallel. This advantage is observed

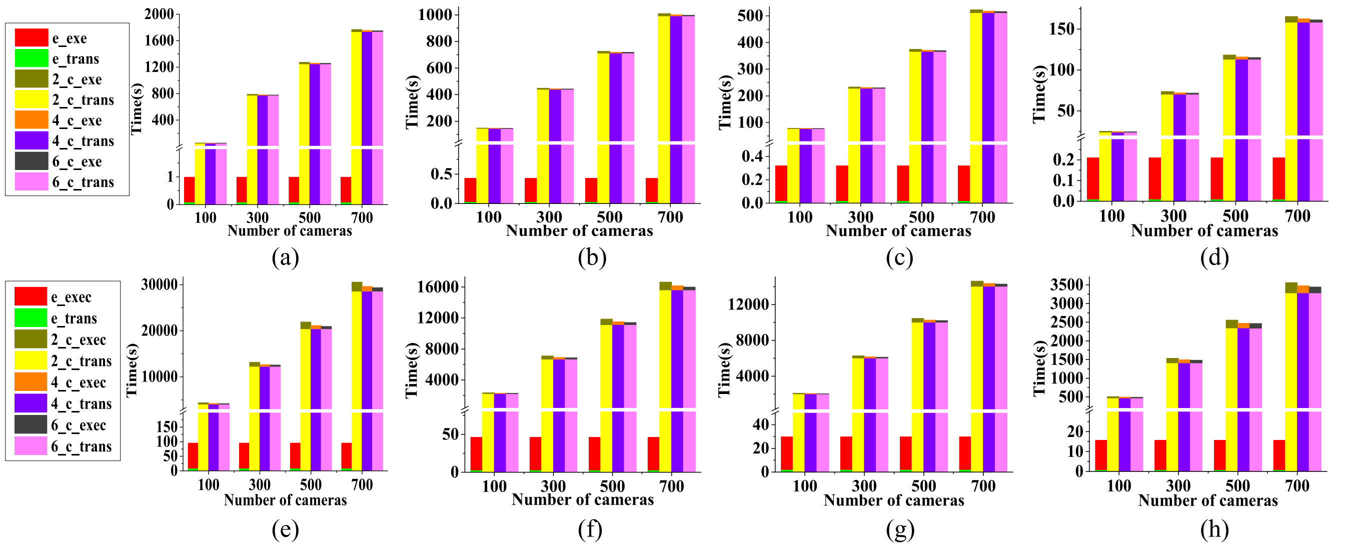


Fig. 7. MTTDs of failures **C** (abnormal brightness) and **J** (flicker screen) in the edge and the cloud models with four resolutions. The result is specified as (Failure type : Resolution) in this figure. For example, (C : 1920 × 1080) denotes the MTTD of the abnormal brightness failure detection under a resolution of 1920 × 1080. The *c* and *e* in *x_**c**_x* and ***e**_x* represent the cloud and edge models, respectively. In the cloud model, the MTTD consists of the transfer time from a camera to the cloud (such as *2_c_trans*) and the execution time (e.g., *2_c_exec*) of failure detection. In the edge model, MTTD consists of the execution time (i.e., *e_exec*) and transfer time (such as *e_trans*); however, the latter is much smaller than the former. In the cloud model, we configure two-processor, four-processor, and six-processor modes and use them to study the MTTDs of failure detection approaches in a cloud server. Notation *4_c_exe* denotes that the testbed is configured with 4 processors for failure detection in the cloud model. (a) C : 1920 × 1080. (b) C : 1280 × 720. (c) C : 720 × 576. (d) C : 352 × 288. (e) J : 1920 × 1080. (f) J : 1280 × 720. (g) J : 720 × 576. (h) J : 352 × 288.

in the image- and video clip-based detection approaches in this model. For example, the MTTD reaches 1769.5 s (30542.4 s), 1753.9 s (29590.1 s), and 1749.2 s (29353.5 s) for image-based (video clip-based) detection using two, four, and six processors, respectively, for 700 data items.

In addition, the transmission and execution time for a failure detection reduces as the resolution decreases. Considering the brightness detection using two processors as an example, the MTTD for the 700-camera data items reduces to 165.4 s (158.4 s for transmission and 7.0 s for execution), 523.2 s (511.5 s for transmission and 11.7 s for execution), 1009.9 s (991.5 s for transmission and 18.4 s for execution), and 1769.5 s (1737.1 s for transmission and 32.4 s for execution) at resolutions of 352 × 288, 720 × 576, 1280 × 720, and 1920 × 1080, respectively. This trend also is also observed in the flicker detection.

D. Scalability Evaluation

We compare the scalability of the VU-based failure detection framework to that of a commercial cloud-based failure detection system. In a real-world video surveillance system, the commercial failure detection system, which is a cloud model, is deployed by Uniview Company at Anhui University. As discussed above, there are more than 4000 cameras on the campus. We create data items from 2000, 3000, and 4000 cameras to evaluate the scalability of VU in the terms of accuracy and MTTD.

We compare the accuracy between the VU and Uniview platforms in Fig. 8(a). In the 2000-camera case, the sum of *TP* and *TN* from detection for six failure (i.e., ***B***, ***C***, ***D***, ***F***, ***H***, and ***I***) is close to 2000 when a detection approach has a high accuracy. In this test, TP_u and TP_{VU} are 324 and 337,

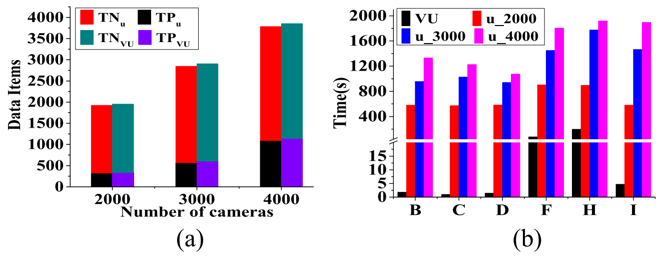


Fig. 8. (a) Accuracy and (b) MTTD for detection approaches in both the VU and Uniview platforms. We select six types of failure detection approaches, i.e., *B* (Blurring), *C* (Abnormal Brightness), *D* (Color Cast), *F* (Freeze Frame), *H* (Occlusion), and *I* (Flicker). TN_u and TN_{VU} represent the number of TN test data items. There are 2000, 3000, and 4000 test data items (cameras). VU denotes the MTTD of our detection approaches in the VU library and *u_2000* is the MTTD of a failure detection in the 2000-camera case.

respectively. Similarly, the value of TN_u is smaller than that in TN_{VU}. (TN + TP) in the VU-based detection system is larger than that in the Uniview platform. Thus, the VU-based detection approaches are more accurate than those in the Uniview platform. The same trend exists in 3000- and 4000-camera cases. To handle video data from cameras with various resolutions, the detection approaches are automatically configured with satisfactory parameter values during failure detection.

In addition, we study the MTTDs of six failure detection approaches, see Fig. 8(b). We compare the longest MTTD of a detection approach in VU with that of the realistic Uniview platform. The longest MTTD of a detection approach for each of the six failures is tested by using the highest-resolution data items that are evaluated in Fig. 7. We find that the longest MTTD for each detection approach is smaller than that in Uniview platform. The VU-based MTTD decreases as

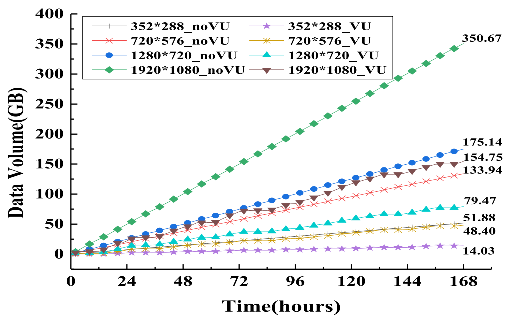


Fig. 9. Costs of video storage usage improvement in the VU-based framework. We employed 168-h (a week) test video data in the experiment. Notation 352 × 288_noVU represents the video data is resolution 352 × 288 and VU-based failure detection framework is not used. While the symbol 352×288_VU means that the VU-based failure detection framework processes resolution 352 × 288-based video.

the solution of an image or video clip decreases. Thus, VU-based detection approaches also have also superior scalability in terms of MTTD compared with the Uniview platform.

E. Storage Usage Improvement in the VU Model

This experiment evaluates the advantages of the VU framework in terms of the improvement of video storage usage in the cloud. In the video surveillance system at Anhui University, we collect one week of continuous video data from a camera at four resolutions (i.e., 352 × 288, 720 × 576, 1280 × 720, and 1920 × 1080). The failure types in the video streams are arbitrary. Without loss of generality, the test data includes five configured types of failure, namely, solid color, blurring, color cast, freeze, and flicker, which are detected via failure detection approaches in the VU-based framework to reduce the video storage cost in the cloud. In addition, we configure two test platforms: 1) with the VU-based framework and 2) without the VU-based framework (*noVU* in short). Useless video data with a low VU value is directly discarded in the first framework but continually stored in the second framework. Both frameworks record the total amount of video data in the cloud every 2 h.

In Fig. 9, the results demonstrate that the noVU-based video data linearly increases with a slope of approximately one over time at the four resolutions. In comparison, the increment trend in VU-based storage usage cost of video data obviously fluctuates and decreases with time due to the discarding of useless video. The VU-based slopes of the curves at the four resolutions change with time. This is attributed to the variable failure data volume with time. The results at the 1920 × 1080 resolution demonstrate that the video data volume at periods of 24 h, 36 h, 72 h, and 84 h increase slightly, while the change in video storage cost is substantial between 96 h and 132 h. For example, the rate of change is close to one. Meanwhile, similar trends are observed at other resolutions. For total video data that are stored in the cloud during 7×24 h, we find that the noVU-based video data volume is 51.88 GB, 133.94 GB, 175.14 GB, and 350.67 GB at resolutions of 352 × 288, 720 × 576, 1280 × 720, and 1920 × 1080, respectively. The VU-based video data volume decreases by up to 63.9%, 64.2%, 54.6%, and 55.9%, respectively, compared to the noVU framework.

From the experimental results, we observe two phenomena.

1) *The Growth Trend of the Video Data Volume in the Cloud:* In the VU-based framework, the growth rate of the storage usage of the data volume in the cloud can vary over time; hence, this framework is considered an elastic storage framework. However, the growth rate of video data in unit time will not exceed that in the noVU-based framework. It can be seen that the network bandwidth that is required for video transmission per unit time will reduce based on the VU framework (see Section VI-F).
2) *The Total Amount of Video Data in the Cloud:* The VU-based total video storage usage costs are substantially reduced, which decreases the storage usage of failure data and improves the utilization of video storage space.

F. Bandwidth Usage Improvement in the VU Model

In Section VI-E, we have found that the VU-based framework filtered useless video data in video streams which is transmitted to the cloud from cameras. This method improves storage utilization in the cloud. It is supposed that the bandwidth cost of video data per unit time reduces. Then, the VU model also saves network bandwidth by preprocessing failure in video streams. This article aims at evaluating the advantages of VU model in terms of bandwidth cost improvement. We select two types of video streams (i.e., failure and normal) with four resolutions (such as 352 × 288, 720 × 576, 1280 × 576, and 1920 × 1080) from the dataset in Section VI-E. The length of each test video streams is 10 min. The failure video streams contain solid color, blurring, color cast, freeze, and flicker. For failure video, there are 3551, 3647, 3090, and 3337 failure frames in the 10-min video streams with 15 000 frames at resolutions of 352 × 288, 720 × 576, 1280 × 576, and 1920 × 1080.

At the edge node, Raspberry Pi3 in Section VI-B2 are employed to capture video streams, perform VU-based failure detection, and push video data to the cloud. The default network bandwidth on RPi3-based platform is 1000 Mb/s and TCP protocol is used in the data transmission. A Dell R730 server in Section VI-B2 is used to receive the video data from the edge node. The data volume transferring to the cloud and the processing time and transfer time on the network are used to calculate the average bandwidth. The results are listed in Table II.

In the noVU-based framework, the bandwidth for video data with normal or failure presents high occupation, such as among 0.8 MB/s ∼ 1.2 MB/s. This reason is that the entire video data is transmitted to the cloud without data processing at the edge, which costs bandwidth in the network. As listed in Table II, the bandwidth for normal data demonstrates a low because the latency of data processing is caused by the module of a failure detection in the VU model. However, the bandwidth exhibits a small change and the impact is minimal, lowing than 0.1 MB/s. For failure video, the VU-based

TABLE II
NETWORK BANDWIDTH IMPROVEMENT IN THE VU-BASED FRAMEWORK

	Normal Case		Failure Case	
	noVU	VU	noVU	VU
352×288	1.042 MB/s	0.998 MB/s	0.825 MB/s	0.519 MB/s
720×576	1.058 MB/s	0.984 MB/s	1.184 MB/s	0.472 MB/s
1280×720	0.962 MB/s	0.893 MB/s	1.142 MB/s	0.424 MB/s
1920×1080	0.953 MB/s	0.854 MB/s	0.859 MB/s	0.477 MB/s

In this experiment, *Normal Case* and *Failure Case* represent the normal and failure video data, respectively. The notations *noVU* and *VU* denote the framework without failure detection approaches and VU-based framework, respectively.

framework reduces the bandwidth cost than that of the noVU-based framework. For example, the bandwidth decreases by approximately 37.1%, 60.1%, 62.9%, and 44.5% at resolutions of 352 × 288, 720 × 576, 1280 × 720, and 1920 × 1080, respectively. The failure detection module in the VU model lengthens the procedure of data processing at an edge node; and then, the average bandwidth cost is around 0.1 MB/s. However, the bandwidth of video streams is improved in the VU model because the failure video data which are detected and discarded by failure detection approaches in the VU, which improves bandwidth in the network.

In summary, the bandwidth cost of useless video data reduces, which largely saves bandwidth in the VU-based video surveillance system. These failure detection modules in the VU framework slightly increase the latency of data processing at the edge, which impairs the real-time in applications. However, the failure detection approaches reduce useless video data and bandwidth cost in the network.

G. FADA Scheduling Scheme in the VU Model

In this section, we perform experiments to validate the benefits of FADA scheduling scheme in VU model. FADA aims at realizing an overall tradeoff among the MTTD, the frequency of failure detection approaches, and the energy consumption (or cost), which are test metrics in this test. According to the details of FADA in Section V-B, the time interval for two applications of the same detection approaches for failure is 10 s (i.e., $T_D = 10$ s) in this test. FADA adjust the detection approach for the failure, which is continually conducted three times, to the first place. Meanwhile, the time interval between two applications of the same failure detection approach is reduced by half, which increases the frequency of failure detection and warns people to handle the failure. Thus, FADA promptly detects failures to shorten the MTTD; however, it increases the energy consumption. Otherwise, FADA enlarges the time interval between the two applications of the same detection procedures if no failure in the video stream. This case reduces the detection frequency and, hence, the energy consumption.

In the experiment, we use three types of test video clips, i.e., normal video clips, failure video clips, and mixed normal and failure video clips. The latter two include five types of failures, i.e., solid color, color cast, blurring, freeze frame, and

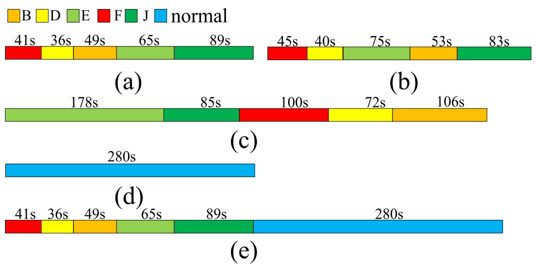


Fig. 10. Test video clips in the evaluation of FADA scheduling scheme. The symbol *fvc* denotes a failure video clips-based test data item with several types of failure. In this test, we configure three types of failure video clips, such as fvc_1, fvc_2, and fvc_3. The *nvc* represents normal video clips without failure. The *mvc* indicates the video clips mixed with normal and failure. (a) fvc_1. (b) fvc_2. (c) fvc_3. (d) nvc. (e) mvc.

TABLE III
EVALUATION ON FADA SCHEDULING SCHEME

	Energy Cost (Joule)		MTTD (seconds)		Frequency	
Data Item	FADA	noFADA	FADA	noFADA	FADA	noFADA
fvc_1	2196	1728	13.61	18.74	43	25
fvc_2	2016	1404	12.01	12.71	43	27
fvc_3	2052	1584	5.88	6.77	64	42
nvc	5112	6588	174.54	162.77	8	11
mvc	7848	8244	42.49	63.26	52	36

FADA refers to the tested results that are obtained using the FADA scheduling scheme in the VU model and noFADA represents the sequential order for failure detection approaches without the FADA scheduling scheme.

flicker, is listed. As listed in Table III, we configure three failure video clips, namely, fvc_1, fvc_2, and fvc_3, see Fig. 10. We configure the test video items with various sizes and failure sequences for the following reasons: 1) solid color, color cast, and blurring are detected by using video clips while failure detection approaches employ images to detect freeze frame and flicker and 2) the scheduling order of failure detection approaches in FADA is determined by the sequence of failure in video streams. Then, the failure time interval in the subsequent procedure of failure detection will be influenced as well.

Fvc_1 with failure video aims at evaluating three test metrics under FADA and noFADA scheduling in the VU model. The results demonstrate that the FADA-based VU improves MTTD and increases energy consumption and the frequency of detection approach. This is because FADA first schedules an approach for a failure that has continually occurred three times. Meanwhile, the time interval for this detection approach will be reduced by half. Then, the frequency of failure detection is increased by FADA to 43 from 25 and the energy consumption increases by up to 27.1%. In addition, the MTTD decreases than that of noFADA. In *nvc* with normal video, FADA decreases the frequency of application of failure detection approaches and energy consumption but increases the MTTD compared with noFADA. If no failure detection approach in the VU model detects a failure in three successive applications, the time interval between the same two approaches is doubled. FADA reduces the frequency of failure detection approaches; thereby enlarging MTTD and lowering energy consumption. For the mixed failure and normal video (i.e., *mvc*), the test metrics depends on the sizes of the normal and failure video clips. The results demonstrate that energy consumption and MTTD in FADA are reduced by approximately 5.0% and 32.8%, respectively, compared to noFADA. The main reason is that FADA schedules detection approaches more frequently for portions of failure video clips compared to the normal video clips. Although the energy cost of failure video in FADA is higher than that in noFADA, the energy cost is much smaller. For the normal-video test, the failure detection approaches are applied fewer times compared to noFADA; however, the energy cost in FADA is much higher than that of noFADA due to the larger number of failure detections. For fvc_1, fvc_2, and fvc_3, we observe that FADA has more failure detections than noFADA, which increases energy consumption but reduces MTTD. Thus, FADA outperforms noFADA on the video stream with various failure sequences.

In summary, the FADA scheduling scheme improves MTTD, the frequency of failure detection approaches and energy consumption under various failure-type and failure-sequence cases, compared to noFADA.

VII. Related Work

In this section, we present related work on: 1) quality of video services, which is evaluated subjectively and objectively; 2) edge computing in video-analytics applications; and 3) this article on VU.

A. Quality of Video Services

Surveillance video data has become the largest source of video data and the amount of available data is growing exponentially in big multimedia data. Video surveillance systems are impacted by distortion or artifacts in the video signal. Video data is lost in transmission. These effects negatively impact the QoS [11] or the QoE [37] of a video data system. Rich literature about the evaluation quality of video services predominately focuses on QoS and QoE. QoS and QoE, between which an interdependent relationship exists, are applied to measure the quality of video services.

QoS is defined by the International Telecommunication Union as the collective effect of service performances, which determine the degree of satisfaction of a user with the service, which is specified in Recommendation E.800 [38]. From the technical perspective, QoS parameters, which are mostly used in the realm of video service, are considered as measurement metrics for video services that are offered by providers. QoS metrics are typically defined in terms of network service metrics, such as the error rate, bit rate, throughput, transmission delay, round trip time, and loss ratio. These video distortions are mostly related to the loss of compression and encoders for video data in edge cameras. Methods of QoS measurement not only require a reference for the rare video data but also use an extra control channel to measure the quality of video service [39]. However, the QoS methods are insufficient for measuring the overall surveillance video quality, because they consider only the packet loss and the delay.

QoE [40] typically considers the quality of video services from the user perspective. QoE evaluations mainly include three types of methods, i.e., subjective, objective, and hybrid. A subjective QoE method [41] is one of the most reliable methods for determining the perception of a user. Several evaluation subjects provide the results of a subjective experiment as the individual scores, which are used to calculate the mean opinion score (MOS) [42] which is specified in ITUT Recommendation P.800 [43] and has emerged as the most popular descriptor for media quality. The subjective approach is characterized by the workforce and time consumptions. This measurement cannot guarantee video services assessment; therefore, this method is not widely used in the measurement of video services. Objective QoE approaches [44] use algorithms to measure the quality of video services by collecting technical parameters from the network. As a result, the evaluation can be performed fast online; however, the human perception could influence the accuracy of the results. Compared with the methods above, a hybrid approach [45], which combines the subjective and objective approaches, assesses QoE fast and reduces the time and human resources costs.

Several approaches to evaluate video quality [41], [46] according to its spatial and temporal characteristics. The former requires too much time and computing resources. The latter is used by the resource-limited edge nodes to preprocess video data. Video quality assessment models [47], [48] are classified into three types, e.g., full-reference (FR), reduced-reference (RR), and no-reference (NR). FR methods [49], [50] compare original images (high quality) to the candidate images whose quality is to be evaluated. This approach must access the original images as references. An edge camera has limited storage space and cannot store entire original images. RR methods [51], [52] use partial features information that is extracted from the original images to evaluate the distorted image. NR methods [53], [54], which do not utilize the original images, use information from the pixel domain and the bitstream of an image/video to perform video quality assessment. These methods perform real-time video quality assessment on a resource-limited platform. The hybrid method for video quality assessment is fully applicable to a wide variety of failures in video clips and images.

B. Edge Computing

Edge computing [55] (which is similar to fog computing [56], [57] and cloudlet [58]) is a new computing model that has emerged with the proliferation of the IoE in recent years. In the IoE era, a huge volume of data will be generated by things that are immersed in our daily lives, and hundreds of applications will be deployed at the edge to consume these data. The term *edge* in edge computing refers to all computing and network resources along the path between data sources and the cloud. Fast online video analysis in a video surveillance system [59], [60] is one of the most prevalent use-cases for edge computing. Cloud computing [61], [62], as the de facto centralized big data processing platform, is insufficient for supporting a video surveillance system in the IoE era due to: 1) the inability of the computing resources that are available in the centralized cloud to keep up with the explosively growing computational requirements of massive video data from edge-based cameras; 2) the longer user-perceived latency that is caused by the data movement between edge cameras and the cloud; 3) the privacy and security concerns of data owners in the edge; and 4) the energy constraints of the edge devices. These issues in the centralized big data processing era have pushed the horizon of a new computing paradigm, namely, edge computing, which calls for processing data at the edge of the network [63].

As discussed above, QoS mostly focuses on video quality from the perspective of video service providers. QoE emphasizes that manual interaction is involved in the evaluation of video quality in the cloud. Hence, large volumes of useless video are transferred to the cloud, which aggravates the network bandwidth. These evaluation criteria are not sufficiently comprehensive for evaluating VU.

This article focuses on: 1) image/video processing and 2) edge computing. For image/video processing, we migrate tasks of image/video processing-based failure detection approaches in the cloud model to the edge nodes in which video failures are detected via the edge computing model. These edge computing-based failure detection approaches realize high real-time performance for video preprocessing, save bandwidth in the network, and reduce the amount of useless video that is stored in the cloud. For edge computing, this article, to the best of our best knowledge, is the first work to employ edge computing in the field of image/video failure detection, the framework of which is called VU. The edge computing paradigm enables edge nodes to perform image/video processing, which improves the performance of failure detection and satisfies real-time demands for large-scale video surveillance systems. Our proposed VU analytics for a large-scale video surveillance system is a well killer application of edge computing [64].

In summary, we propose a VU framework for analyzing video quality and usefulness in an unmanned mode on the fly. VU, which combines video/image processing and edge computing, employs edge computing to preprocess video streams and to detect failures in the video on edge nodes. VU reduces the bandwidth of video transmission to the cloud and optimizes video storage in the cloud.

VIII. Discussion

The proposed VU model is derived from and inspired by useless video streams in large-scale video surveillance systems. The objectives of the VU model include: 1) early detecting failures in any part of a video surveillance systems on the fly via video analytics; 2) promptly filtering video data with failures to reduce the network bandwidth overload and improve the storage usage in the cloud; and 3) efficiently and accurately analyzing useful video streams using suitable resources. Therefore, the proposed model is easily adjusted for other video systems, since the three domains are highly generic. The detection methods with satisfactory applicability and feasibility are still employed in other video systems, which have cameras, end-users, network, or cloud servers.

In this article, we only apply the VU model to video surveillance systems. The application that we consider is one of the populations. In addition, we believe that VU model and detection approaches for failures in video streams will be helpful in other video service systems, such as the vehicle video recorder in a general or intelligent connected vehicle system.

In this article, we implemented VU framework that uses two types of computing resources: 1) additional computing resources for edge nodes (e.g., camera or router) and 2) local computing resources in the end-user and the cloud server. We employed and designed several lightweight image processing algorithms, which require low computing resources yet fulfill the accuracy requirements for failure detection. Therefore, we employed a Raspberry Pi V3 platform as the computing unit in video processing for an edge camera. Our VU framework can also be implemented by using deep learning-based methods on high-computing resources platforms (e.g., GPU [24], FPGA [65], and AI chip [66]). These methods are commonly used for computing-intensive tasks, which require high-power consumption. Deep learning methods that are based on Jetson boards may be suitable for applications of object detection algorithms, recognition, behavior analytics, and tracking. In addition, similar and extended approaches, e.g., deep learning methods, which are outside the scope of this article, will be further studied for more sophisticated applications in our future work.

IX. Conclusion

In this article, we proposed a video usefulness model, namely, the VU model, for large-scale video surveillance systems, that is based on edge computing and cloud computing. The VU model aims at not only effectively using the large-scale video data to detect failures that are distributed among edge cameras, end-users, the cloud, and the network but also managing large-scale video surveillance systems. According to the VU model, we summarize three types of failure domains in which numerous failures occur, which are used to evaluate the usefulness of video data. In our experiments, the VU model is evaluated in the following two aspects. First, these fast online failure detection approaches, which are based on edge computing or cloud computing, efficiently improve the MTTD. Second, the most useless video data (e.g., video data with a black screen) is directly handled by the edge devices rather than being uploaded to the cloud. Thus, the bandwidth of the network will not be wasted, and storage savings will be realized in the cloud.

Acknowledgment

The authors would like to thank the reviewers for their work on this article.

References

[1] T. D. Räty, “Survey on contemporary remote surveillance systems for public safety,” *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 5, pp. 493–515, Sep. 2010.

[2] A. Kumbhar, F. Koohifar, I. Güvenç, and B. Mueller, "A survey on legacy and emerging technologies for public safety communications," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 97–124, 1st Quart., 2017.

[3] S. W. Smith, "Video surveillance system," U.S. Patent 6 757 008, Jun. 29, 2004.

[4] T. Huang, "Surveillance video: The biggest big data," *Comput. Now*, vol. 7, no. 2, pp. 82–91, 2014.

[5] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 620–636, Jul. 2003.

[6] M. Uhrina, J. Frnda, L. Sevcik, and M. Vaculik, "Impact of H.264/AVC and H.265/HEVC compression standards on the video quality for 4K resolution," *Adv. Elect. Electron. Eng.*, vol. 12, no. 4, pp. 905–908, 2014.

[7] N. Dimitrova, H.-J. Zhang, B. Shahraray, I. Sezan, T. Huang, and A. Zakhor, "Applications of video-content analysis and retrieval," *IEEE Multimedia*, vol. 9, no. 3, pp. 42–55, Jul./Sep. 2002.

[8] K. A. Joshi and D. G. Thakore, "A survey on moving object detection and tracking in video surveillance system," *Int. J. Soft Comput. Eng.*, vol. 2, no. 3, pp. 44–48, 2012.

[9] B. Yogameena and S. P. Krishraj, "Synoptic video based human crowd behavior analysis for forensic video surveillance," in *Proc. Int. Conf. Adv. Pattern Recognit.*, 2015, pp. 1–6.

[10] M. R. Future. *Video Surveillance Storage Market Research Report—Forecast to 2023*. Accessed: Feb. 10, 2019. [Online]. Available: https://www.marketresearchfuture.com/reports/video-surveillance-storage-market-5848

[11] M. S. Hossain, "QoS-aware service composition for distributed video surveillance," *Multimedia Tools Appl.*, vol. 73, no. 1, pp. 169–188, 2014.

[12] M. Li and C.-Y. Lee, "A cost-effective and real-time QoE evaluation method for multimedia streaming services," *Telecommun. Syst.*, vol. 59, no. 3, pp. 317–327, 2015.

[13] T. T. Collipi and D. F. Harvey, "Method and apparatus for analyzing surveillance systems using a total surveillance time metric," U.S. Patent 7 436 295, Oct. 14, 2008.

[14] T. Kon, N. Uchida, K. Hashimoto, and Y. Shibata, "Evaluation of a seamless surveillance video monitoring system used by high-speed network and high-resolution OMNI-directional cameras," in *Proc. IEEE Int. Conf. Netw. Inf. Syst.*, 2012, pp. 187–193.

[15] A. Osuna, "IBM system storage N series and digital video surveillance," *Adv. Mater. Res.*, vols. 953–954, pp. 1113–1116, 2010.

[16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[17] Q. Zhang, Z. Yu, W. Shi, and H. Zhong, "Demo abstract: EVAPS: Edge video analysis for public safety," in *Proc. Edge Comput.*, 2016, pp. 121–122.

[18] N. Chen, Y. Chen, Y. You, H. Ling, P. Liang, and R. Zimmermann, "Dynamic urban surveillance video stream processing using fog computing," in *Proc. IEEE 2nd Int. Conf. Multimedia Big Data*, 2016, pp. 105–112.

[19] N. Chen, Y. Chen, S. Song, C.-T. Huang, and X. Ye, "Poster abstract: Smart urban surveillance using fog computing," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2016, pp. 95–96.

[20] S. Li, L. Da Xu, and S. Zhao, "The Internet of Things: A survey," *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, 2015.

[21] A. B. Mutiara. *Internet of Things/Everythings (IoT/E)*. Accessed: Feb. 10, 2019. [Online]. Available: https://www.researchgate.net/publication/292607382_Internet_of_ThingsEverythings_IoTE

[22] R. Pi. (2017). *Compute Module Development Kits Now Available!* Accessed: Dec. 14, 2018. [Online]. Available: https://www.raspberrypi.org/blog/compute-module-development-kits-now-available/

[23] Intel. (2018). *Intel®Movidius™ Neural Compute Stick*. Accessed: Dec. 14, 2018. [Online]. Available: https://software.intel.com/en-us/movidius-ncs

[24] NVIDIA. (2017). *Nvidia Jetson Systems—The AI Solution for Autonomous Machines*. Accessed: Dec. 14, 2018. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[25] L. Sendur and I. W. Selesnick, "Bivariate shrinkage with local variance estimation," *IEEE Signal Process. Lett.*, vol. 9, no. 12, pp. 438–441, Dec. 2002.

[26] T. Barbu, "Variational image denoising approach with diffusion porous media flow," *Abstract Appl. Anal.*, vol. 2013, no. 1, p. 8, 2013.

[27] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA, USA: Addison-Wesley, 2010.

[28] M. A. Khanesar, E. Kayacan, M. Teshnehlab, and O. Kaynak, "Analysis of the noise reduction property of type-2 fuzzy logic systems using a novel type-2 membership function," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 5, pp. 1395–1406, Oct. 2011.

[29] J. Gao and N. Liu, "An improved adaptive threshold canny edge detection algorithm," in *Proc. Int. Conf. Comput. Sci. Electron. Eng.*, vol. 1, 2012, pp. 164–168.

[30] H. Dong and D. Y. Han, "Research of image matching algorithm based on surf features," in *Proc. Int. Conf. Comput. Sci. Inf. Process.*, 2012, pp. 1140–1143.

[31] M. Kohn, "Method for analyzing images and for correcting the values of video signals," U.S. Patent 6 683 982, Jan. 27, 2004.

[32] N. S. P. Kong and H. Ibrahim, "Color image enhancement using brightness preserving dynamic histogram equalization," *IEEE Trans. Consum. Electron.*, vol. 54, no. 4, pp. 1962–1968, Nov. 2008.

[33] Á. Bayona, J. C. SanMiguel, and J. M. M. Sanchez, "Comparative evaluation of stationary foreground object detection algorithms based on background subtraction techniques," in *Proc. IEEE Int. Conf. Adv. Video Signal Based Surveillance*, 2009, pp. 25–30.

[34] J. Tang, "A color image segmentation algorithm based on region growing," in *Proc. Int. Conf. Comput. Eng. Technol.*, vol. 6, 2010, pp. 634–637.

[35] J. N. Sarvaiya, S. Patnaik, and S. Bombaywala, "Image registration by template matching using normalized cross-correlation," in *Proc. Int. Conf. Adv. Comput. Control Telecommun. Technol.*, 2009, pp. 819–822.

[36] J. H. Saltzer and M. F. Kaashoek, *Principles of Computer System Design: An Introduction*. Burlington, MA, USA: Morgan Kaufmann, 2009.

[37] P.-H. Wu, J.-N. Hwang, J.-Y. Pyun, K.-M. Lan, and J.-R. Chen, "QoE-aware resource allocation for integrated surveillance system over 4G mobile networks," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2012, pp. 1103–1106.

[38] "General aspects of quality of service and network performance in digital network, including ISDN," ITU, Geneva, Switzerland, ITU-Recommendation I.350, 1993.

[39] M. Vranješ, S. Rimac-Drlje, and K. Grgić, "Review of objective video quality metrics and performance comparison using different databases," *Signal Process. Image Commun.*, vol. 28, no. 1, pp. 1–19, 2013.

[40] K. Piamrat, C. Viho, J.-M. Bonnin, and A. Ksentini, "Quality of experience measurements for video streaming over wireless networks," in *Proc. Int. Conf. Inf. Technol. New Gener.*, 2009, pp. 1184–1189.

[41] T. Tominaga, T. Hayashi, J. Okamoto, and A. Takahashi, "Performance comparisons of subjective quality assessment methods for mobile video," in *Proc. Int. Workshop Qual. Multimedia Exp.*, 2010, pp. 82–87.

[42] R. C. Streijl, S. Winkler, and D. S. Hands, "Mean opinion score (MOS) revisited: Methods and applications, limitations and alternatives," *Multimedia Syst.*, vol. 22, no. 2, pp. 213–227, 2016.

[43] "Mean opinion score (MOS) terminology," Int. Telecommun. Union, Geneva, Switzerland, ITU-Recommendation 800.1, 2003.

[44] Q. Huynh-Thu and M. Ghanbari, "The accuracy of PSNR in predicting video quality for different video scenes and frame rates," *Telecommun. Syst.*, vol. 49, no. 1, pp. 35–48, 2012.

[45] S. Winkler and P. Mohandas, "The evolution of video quality measurement: From PSNR to hybrid metrics," *IEEE Trans. Broadcast.*, vol. 54, no. 3, pp. 660–668, Sep. 2008.

[46] M. Roitzsch and M. Pohlack, "Video quality and system resources: Scheduling two opponents," *J. Vis. Commun. Image Represent.*, vol. 19, no. 8, pp. 473–488, 2008.

[47] Z. Wang, G. Wu, H. R. Sheikh, E. P. Simoncelli, E.-H. Yang, and A. C. Bovik, "Quality-aware images," *IEEE Trans. Image Process.*, vol. 15, no. 6, pp. 1680–1689, Jun. 2006.

[48] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective video quality assessment methods: A classification, review, and performance comparison," *IEEE Trans. Broadcast.*, vol. 57, no. 2, pp. 165–182, Jun. 2011.

[49] T. N. Pappas, R. J. Safranek, and J. Chen, "Perceptual criteria for image quality evaluation," in *Handbook of Image & Video Processing*. Amsterdam, The Netherlands: Elsevier, 2000, pp. 669–684.

[50] K. Manasa and S. S. Channappayya, "An optical flow-based full reference video quality assessment algorithm," *IEEE Trans. Image Process.*, vol. 25, no. 6, pp. 2480–2492, Jun. 2016.

[51] Z. Wang, H. R. Sheikh, and A. C. Bovik, "Objective video quality assessment," *Handbook Video Databases Design Appl.*, vol. 17, no. 5, pp. 1041–1078, 2003.

[52] C. G. Bampis, P. Gupta, R. Soundararajan, and A. C. Bovik, "SpEED-QA: Spatial efficient entropic differencing for image and video quality," *IEEE Signal Process. Lett.*, vol. 24, no. 9, pp. 1333–1337, Sep. 2017.

[53] H. R. Sheikh, A. C. Bovik, and L. Cormack, "No-reference quality assessment using natural scene statistics: JPEG2000," *IEEE Trans. Image Process.*, vol. 14, no. 11, pp. 1918–1927, Nov. 2005.

[54] M. T. Vega, C. Perra, F. De Turck, and A. Liotta, "A review of predictive quality of experience management in video streaming services," *IEEE Trans. Broadcast.*, vol. 64, no. 2, pp. 432–445, Jun. 2018.

[55] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[56] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[57] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, 2015, pp. 37–42.

[58] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *J. Netw. Comput. Appl.*, vol. 59, pp. 46–54, Jan. 2016.

[59] H. D. Park, O.-G. Min, and Y.-J. Lee, "Scalable architecture for an automated surveillance system using edge computing," *J. Supercomput.*, vol. 73, no. 3, pp. 926–939, 2017.

[60] A. Chowdhery, P. Bahl, and T. Zhang, "Bandwidth efficient video surveillance system," U.S. Patent 20 170 078 626, Mar. 2017.

[61] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008. Accessed: Feb. 10, 2019. [Online]. Available: http://doi.acm.org/10.1145/1364782.1364786

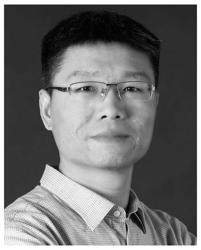
[62] M. Armbrust *et al.*, "A view of cloud computing," *Int. J. Comput. Technol.*, vol. 4, no. 2, pp. 50–58, 2013.

[63] M. Satyanarayanan *et al.*, "Edge analytics in the Internet of Things," *IEEE Pervasive Comput.*, vol. 14, no. 2, pp. 24–31, Apr./Jun. 2015.

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

[65] Y. Shan, "ADAS and video surveillance analytics system using deep learning algorithms on FPGA," in *Proc. IEEE 28th Int. Conf. Field Program. Logic Appl. (FPL)*, 2018, pp. 4465–4650.

[66] I. Stoica *et al. A Berkeley View of Systems Challenges for AI.* Accessed: Feb. 10, 2019. [Online]. Available: https://arxiv.org/pdf/1712.05855.pdf



Hui Sun received the Ph.D. degree from Huazhong University Science and Technology, Wuhan, China, in 2014.

He is an Assistant Professor of computer science with Anhui University, Hefei, China. His current research interests include computer systems, edge computing, performance evaluation, nonvolatile memory-based storage systems, file systems, and I/O architectures.

Weisong Shi (F'16) received the B.S. degree in computer engineering from Xidian University, Xi'an, China, in 1995, and the Ph.D. degree in computer engineering from the Chinese Academy of Sciences, Beijing, China, in 2000.

He is a Charles H. Gershenson Distinguished Faculty Fellow and a Professor of computer science with Wayne State University, Detroit, MI, USA. His current research interests include edge computing, computer systems, energy-efficiency, and wireless health.

Prof. Shi was a recipient of the National Outstanding Ph.D. Dissertation Award of China and the NSF CAREER Award. He is an ACM Distinguished Scientist.

Xu Liang was born in 1994. She is currently pursuing the M.S. degree with Anhui University, Hefei, China.

Her current research interests include computer systems, edge computing, failure detection, and video surveillance systems.

Ying Yu was born in 1996. She is currently pursuing the M.S. degree with Anhui University, Hefei, China.

Her current research interests include edge computing, computer systems, intelligent video analytics, and storage systems.