7-13-2020

A Real-Time Feature Indexing System on Live Video Streams

Aditya Chakraborty
Purdue University

Akshay Pawar
Purdue University

Hojoung Jang
Purdue University

Shunqiao Huang
Purdue University

Sripath Mishra
Purdue University

See next page for additional authors

Recommended Citation

Aditya Chakraborty, Akshay Pawar, Hojoung Jang, Shunqiao Huang, Sripath Mishra, Shuo-Han Chen, Yuan-Hao Chang, George K. Thiruvathukal, Yung-Hsiang Lu, A Real-Time Feature Indexing System on Live Video Streams, Proceedings of COMPSAC 2020.

A Real-Time Feature Indexing System on Live Video Streams

Aditya Chakraborty, Akshay Pawar, Hojoung Jang, Shunqiao Huang, Sripath Mishra, Shuo-Han Chen*, Yuan-Hao Chang*, George K. Thiruvathukal†‡, Yung-Hsiang Lu
School of Electrical and Computer Engineering, Purdue University, USA
*Institute of Information Science, Academia Sinica, Taipei, Taiwan
†Department of Computer Science, Loyola University Chicago, USA
‡Argonne National Laboratory, USA
e-mail: {chakra17, pawar4, jang88, huang953, mishra60, yunglu}@purdue.edu,
*{qoolili, johnson}@iis.sinica.edu.tw, †gkt@cs.luc.edu

***Abstract*—Most of the existing video storage systems rely on offline processing to support the feature-based indexing on video streams. The feature-based indexing technique provides an effective way for users to search video content through visual features, such as object categories (e.g., cars and persons). However, due to the reliance on offline processing, video streams along with their captured features cannot be searchable immediately after video streams are recorded. According to our investigation, buffering and storing live video steams are more time-consuming than the YOLO v3 object detector. Such observation motivates us to propose a real-time feature indexing (RTFI) system to enable instantaneous feature-based indexing on live video streams after video streams are captured and processed through object detectors. RTFI achieves its real-time goal via incorporating the novel design of metadata structure and data placement, the capability of modern object detector (i.e., YOLO v3), and the deduplication techniques to avoid storing repetitive video content. Notably, RTFI is the first system design for realizing real-time feature-based indexing on live video streams. RTFI is implemented on a Linux server and can improve the system throughput by upto 10.60x, compared with the base system without the proposed design. In addition, RTFI is able to make the video content searchable within 20 milliseconds for 10 live video streams after the video content is received by the proposed system, excluding the network transfer latency.**

***Index Terms*—object detection, data storage, real-time, feature-based indexing**

I. Introduction

Feature-based indexing techniques have been widely regarded as an efficient method to access video storage systems and allow users to search the video content through visual features, including object categories (*e.g.*, cars and persons), color, or shape [20]. To support feature-based indexing, video storage systems can be equipped with an object detector to identify and label video streams with identified object categories. Most of existing video storage systems use offline processing to perform object detection after the video is recorded. This offline approach also prevents video storage systems from enabling instantaneous feature-based indexing on live video streams. The ability to perform instantaneous feature-based indexing has become more and more feasible, as modern object detectors are capable of identifying objects in near real-time with the growing speed and accuracy. However, the storage requirements for supporting instantaneous feature-based indexing receive much less attention. Additionally, previous study [7] also suggests that the storage problem could become a major bottleneck as the size of video data continues to grow. Our experiment suggests that buffering and storing video streams consume more time than an object detector (*i.e.*, YOLO v3). To resolve above issues, this study presents a real-time feature indexing (RTFI) system to enable feature-based indexing on live video streams. Please note that the term "real-time" means that the latency between capturing the video content and making it searchable within only 20 milliseconds on average and shorter than the interval between two frames (*i.e.*, 33 ms). The main goal of this study is to support real-time feature indexing by enhancing the efficiency of storing and indexing live video streams for bridging the gap between modern object detectors and storage systems. The technical difficulty of this study lies in *how to make the live video streams searchable with features in real-time after the video content is received by the proposed system.*

Feature-based indexing, also known as content-based indexing, can search video content conveniently via features, such object categories, within the videos. For instance, after the video content is processed through an object detector, feature-based indexing allows users to search all video frames consisting of cars in video clips. Thus, users do not need to go through the whole video clips to find video frames consisting of cars. To support feature-based indexing, various designs have been proposed. For instance, Lyer et al. [8] proposed the auto-annotation of videos. However, the design goals of previous studies are to enable efficient offline feature-based indexing after the video clip is captured and stored. For example, previous studies [8, 10] mainly rely on the conventional database or file systems directly. The storage system designs, including the design of metadata structure and

data placement, to enable instantaneous feature-based indexing for live video streams have received much less attention.

The need for supporting instantaneous feature-based indexing for live video streams has grown rapidly, owing to the growing speed and accuracy of modern object detectors. For instance, convolution neural networks (CNNs)-based object detection has been widely regarded as an effective method to identify objects. Various CNN-based object detectors, such as Faster-RCNN [16] and YOLO [14], have been investigated. Among these neural network designs, YOLO has gained popularity since 2016 due to its high speed and has released its third version [15] in 2018. YOLO v3 has improved on its previous accuracy by 1.5x and maintained the speed of detection around 20 FPS. With these improvements, YOLO v3 can preform real-time object detection with its high speed and has been used in several fields such as self-driving cars [6], surveillance systems [5], and robotics [19]. Meanwhile, to achieve low latency in object detection, numerous studies have also been proposed. For example, Luo et al. [13] propose a scheduler network to formulate the object detector problem as a sequential decision problem. Most of the previous studies focused on improving the efficiency and performance of object detector; *the storage requirements of storing and indexing video content with their identified object categories are neglected.*

This paper presents a real-time feature indexing (RTFI) system to support image and video retrieval with specified object categories in a short latency after the video is captured. Notably, object categories are referred to as features in the rest of this paper. The proposed RTFI system includes three major components: (1) similarity deduplication module, (2) feature-aware storage allocator, and (3) feature-as-metadata indexing scheme. In the proposed system, live video streams can be captured through public surveillance cameras and arrive on the proposed system through network connections. These video streams are then passed through a similarity deduplication module to skip video frames that have high similarity to the previous frames for avoiding repeatedly processing and storing similar frames. Next, the deduplicated video content is passed through the YOLOv3 object detector to extract a set of predefined objects (*e.g.* cars and persons). Note that this study refers to YOLOv3 as the object detector for retaining the excellence of previous studies and focus on the storage system design. Next, to facilitate the read performance of video frames with different features, video frames containing the same feature are stored on the adjacent storage space for enhancing the spatial locality and reduce access latency. Finally, to achieve the goal of lower latency retrieval, the feature-as-metadata indexing scheme is included to search video with certain features with a short latency. The proposed system is implemented on a Linux v4.15 server and tested with 10 live video streams. We then measure the time interval between a video frame is received by RTFI and a video frame is stored with its identified object categories. Evaluation results show that the proposed RTFI could improve the system throughput by upto 10.60x, compared with the base system without the proposed design.

The contributions of this study can be listed as follows.

- This study is a pioneer system design proposed to enable efficient feature indexing based on object detection for live video streams in real-time.
- The proposed RTFI boosts the efficiency of feature-based indexing on live video streams via eliminating frames with high similarity, allocating storage space with feature-awareness, and managing stored video frames through identified object types.
- Experimental results show that the proposed RTFI could achieve upto 70 frame per second (FPS) with multi-process implementation and upto 10.60x system throughput improvement when compared with the base system.

The rest of this paper is organized as follows. Section II discusses the background and the motivation for conducting this research. Then, Section III describes the details of the proposed RTFI system. Next, Section IV reports the experimental results. Finally, Section V concludes this paper.

II. Background and Observation

A. Video Storage for Feature-based Indexing

Most of the existing feature-based indexing designs rely on the conventional database and file systems for storing the video content and their features. However, these database and file systems are usually not optimized for storing video clips. For instance, the default file system for the Linux system is a block-based file system, which has been proven less suitable for unstructured data, such as videos and images [18]. Meanwhile, object storage stores the data content and its metadata as an object and is considered to be more suitable for unstructured data. In other words, most of the existing feature-based indexing designs do not consider the characteristics of the stored video content and the organization of feature entries during the design phase.

In addition, previous works mainly focused on analyzing the performance difference between block-based file systems and databases without proposing a storage design to support the feature-based indexing within a large dataset. For instance, Lim et al. [11] compared the performance of CNN-based object detectors when the dataset is stored in block-based file systems. According to their investigations, storing datasets directly on block-based file systems can result in up to 17 times slower speed for the CNN-based object detector. This is because the metadata operations of block-based file systems require long processing time to access the storage devices. Nevertheless, the efficiency problem of feature-based indexing for large datasets is still neglected. Such observation motivates us to *explore the possibility of enabling a real-time feature indexing system via object storage and database systems.*

B. CNN-based Object Detection

In the field of object detection, the convolution neural network (CNN) has gained its popularity rapidly with its huge success and high efficiency in the field of image processing. CNN is typically composed of an input layer, convolutional layers, pooling layers, fully-connected layers, and an output layer. Within each layer, there are one or multiple neurons, which are connected to neurons in the next layer. The models of CNN-based object detection can be divided into two categories. One of them is known as the two-stage model, which utilizes the region proposal method to perform the region of interest (ROI) selection during the detection. One example of the region-proposal-based CNNs is Faster-RCNN [16]. However, the two-stage model is less preferable for low latency object detection due to their relatively longer inference latency. For instance, as summarized in Table I, Fast-RCNN has the longest latency for performing inference on one single image. Note that the term "inference" refers to perform object detection with the trained CNN model.

On the other hand, another category of CNN-based object detection is the one-stage models, such as YOLO and RetinaNet [12]. Instead of using the region proposal method, the one-stage models directly detect the objects. For instance, YOLO simultaneously predicts multiple bounding boxes and classification class probabilities. On the other hand, RetinaNet uses ResNet for deep feature extraction and feature pyramid network for constructing a multi-scale feature pyramid that is used for detection.

TABLE I
COMPARISON OF CNN-BASED OBJECT DETECTION [15].

Mode	mAP	Inference Latency (ms)
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198
Faster R-CNN+++	34.9	3360
YOLOv3 416×416	31.0	29
YOLOv3 608×608	33.0	51

As shown in the Table I, RetinaNet-101-800, which is built on a ResNet-101 feature extractor with an input shape of 800×800, can achieve the highest accuracy with a slightly longer latency than YOLO v3. On the other hand, YOLO v3 model with a 608×608 input layer can achieve the mean average precision (mAP) of 33.0, which is comparable to that of RetinaNet-101-800, while the inference latency of YOLOv3 is almost 4 times shorter than that of RetinaNet-101-800. mAP is a metric in measuring the accuracy of object detectors and the higher mAP means the object detector is more accurate. Meanwhile, although two-stage object detection methods, such as Faster R-CNN, can achieve slightly higher accuracy than YOLOv3, their inference latency is much longer than that of YOLOv3. Therefore, the YOLOv3 is chosen as the object detector of this study. Although both the accuracy and inference speed of CNN-based object detector have been studied extensively, *the data storage requirements for storing the huge amount of feature-extracted video receives much less attention.*

C. Observation

With the growing capability of object detectors, modern video storage systems (*i.e.,* automated surveillance system [4, 9, 17]) can identify the objects in each video frame accurately. Nevertheless, due to the performance mismatch between existing block-based storage systems and CNN-based object detection, the efficiency of storing and indexing video clips with specified object categories after CNN inference is degraded. In addition, most of the previous studies focus on improving the efficiency and performance of CNN-based object detector and few of them have looked into the design of enabling efficient storing and indexing of video content with identified features. To support storing and indexing live video streams with their inference results, one viable option is to utilize existing object storage and database systems directly. However, the main challenge is that the hard disk drives can only sustain a certain amount of write traffic per second. In other words, after the maximum allowance of write traffic, the efficiency of storing and indexing degrades drastically.

To study the latency of processing and storing live video streams, a preliminary experiment has been conducted. The composition of latency induced by buffering video streams on storage, performing object detection with YOLO v3, and storing video content with their identified object categories is summarized in Figure 1. Notably, this experiment is conducted on a MySQL-based database (*i.e.*, Vitness [3]), an object storage system (*i.e.*, Minio [1]), and a solid-state drive (SSD) (see Table II). Based on this configuration, the results suggested that buffering and storing live video streams is actually more time consuming than the YOLO v3 object detector.

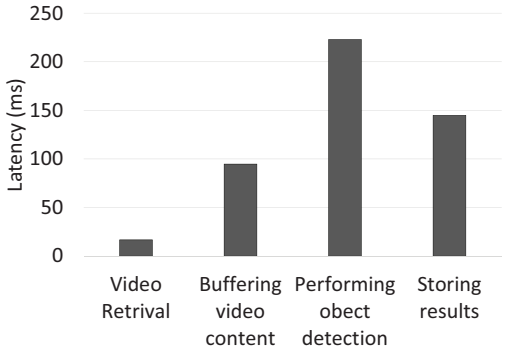


Fig. 1. The composition of latency induced by each component during performing object detection and storing the video content with their identified object categories.

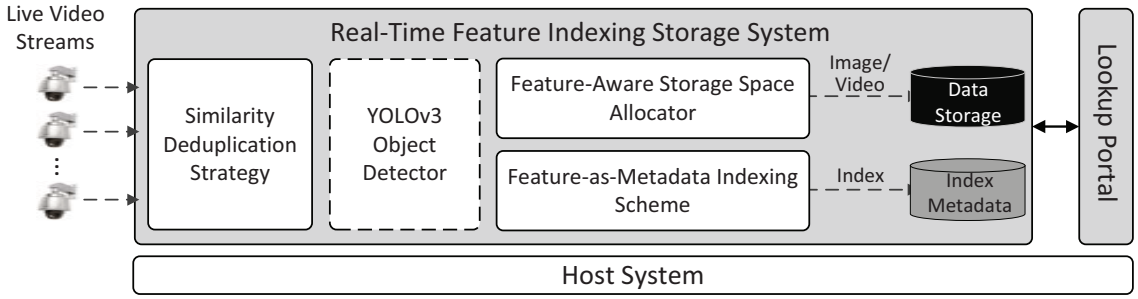


Fig. 2. The architecture of RTFI system, which includes the similarity deduplication module, object detector, feature-aware storage allocator, and feature-as-metadata indexing scheme to enable instantaneous feature-indexing on live video streams.

Even though utilizing other storage devices with higher write performance can partially resolve the performance issue, the storage performance bottleneck remains as the data size of live video streams grows. Storage performance bottleneck prevents feature-based indexing from being real-time, owing to the high latency of storage systems. Thus, the major technical problem is how to provide efficient storing and indexing on a large number of live video streams with low latency after the video content is processed by CNN-based object detectors. To resolve this issue, we propose the real-time feature indexing (RTFI) system to efficiently conduct object detection on incoming live video streams and make those video clips searchable with identified object categories (*e.g.,* persons and cars) in an efficient manner with the focus on reducing latency and redundancy.

III. Real-Time Feature Indexing System

A. Overview

To enhance the efficiency of storing and indexing live video streams, the study presents a real-time feature indexing (RTFI) system for performing object detection and making the inference results searchable with a short latency. To the best of our knowledge, *this study is the first system design for realizing real-time feature indexing on live video streams with their identified object categories.* The main methodology of this study is to avoid passing similar video frames through a CNN-based object detector and store video frames. Therefore, the latency of storing a video frame and making it searchable based on object detection results can be lowered. To enforce the aforementioned main methodology, the proposed RTFI system is composed of three main components along with the YOLOv3 object detector.

The system architecture of the RTFI system can be summarized as Figure 2. As shown in the figure, the RTFI system first utilizes the similarity deduplication module to eliminate those video frames that have high similarity when compared with previously stored frames (see Section III-B). After the similarity deduplication module discards similar frames, the rest of the video frames are passed through the pre-trained YOLOv3 object detector to extract the object detection results. Next, the Minio object storage is utilized as the building block along with the proposed feature-aware storage allocator for reducing the retrieval latency of video clips with a specific object category (see Section III-C). Notably, Minio is an object storage server, which is suitable for storing unstructured data such as images, videos, log files, archives, containers, etc. Unlike files in a filesystem, objects are stored in flat structures with no folders, directory, or hierarchy. Finally, the RTFI system manages and indexes those stored video frames with the feature-as-metadata indexing scheme (see Section III-D), which is based on the Vitess database to facilitate the management of indexes with atomicity, consistency, isolation, and durability (ACID) guarantees. Notably, *directly applying Minio and Vitess for indexing video content through object categories is inefficient because the bottleneck of underlying storage devices still exists.* Therefore, the proposed RTFI system aims to resolve the bottleneck of underlying storage devices while enabling efficient video content retrieval with specified features with the proposed components.

B. Similarity Deduplication Module

Because storing latency is an important factor in the proposed RTFI system, having the ability to eliminate unnecessary object detection latency and read/write latency of the storage devices is critical. To eliminate those unnecessary latency, the RTFI system employs the similarity deduplication module to avoid processing video frames that have high similarity to the previously stored frame. To compare two video frames, the included module performs the structural similarity (SSIM) [21] comparison to determine the similarity between two video frames via the luminance, contrast intensity, and local structure of the video frames. In this study, the output of the SSIM method is normalized between 0 and 1 to represent the similarity. If the output value is 1, it means that the two input frames are identical. On the other hand, the output value of 0 means that the two input frames are completely different. Then, the SSIM output value is compared with a predetermined threshold to determine whether the incoming

video frames should be sent to the object detector and to be stored. Notably, the predetermined threshold is set to 0.85 based on our investigation.

In addition to eliminating the video frames with high similarity, the included module also avoids skipping too many frames by setting the storing interval between two stored video frames of a video stream. The storing interval is set to 0.1 seconds initially for each video stream and is adjusted constantly according to the similarity patterns of each video stream. The adjustment is required because the content of video streams could show varying similarity results and utilizing a single interval for all video streams is not ideal. For example, video frames from a live video stream of a crowded subway station show lower similarity than that of a live video stream recording the top of a less-traveled mountain road. Based on the similarity and interval check, the workflow of the similarity deduplication module can be shown as Figure 3.

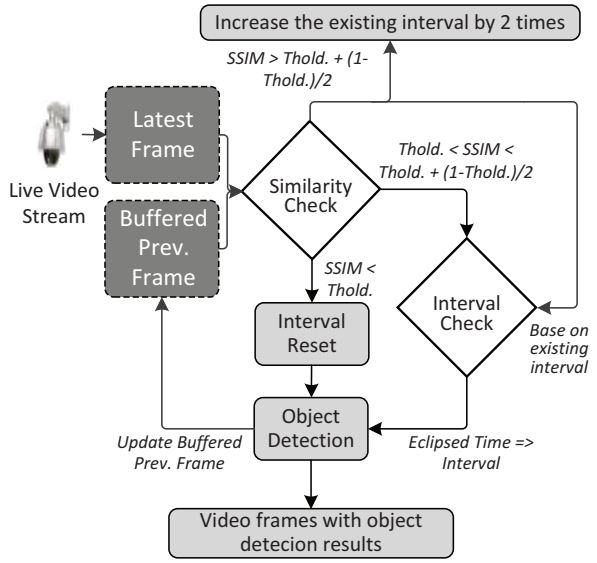


Fig. 3. The similarity deduplication module, which is utilized to remove video frames with high similarity for avoid processing the similar video frames for multiple times.

As shown in Figure 3, after the latest video frame is captured from the live video streams, it is sent to the similarity check component with the previous frame for the SSIM comparison. If the output SSIM value is higher than the $threshold + (1 - threshold)/2$, it means that changes in these two video frames are almost non-existent. $threshold+(1-threshold)/2$ gives the midpoint between 1 and the similarity threshold. This study anticipates that such a midpoint value is necessary for finding complete inactivity in video streams. In this case, the included module performs the interval check based on its existing interval value and increases the storing interval to 2 times of its existing value. By increasing the interval, the included module can further avoid storing frames with high similarity when the interval expires. On the other hand, if the output SSIM value is smaller than the threshold, the incoming frame will be sent to the object detector and the buffered previous frame will be updated to the incoming frame. In addition, the storing interval is reset to its initial value to avoid losing video frames that could contain different objects.

Finally, the third case is that the SSIM value is larger than $threshold+(1-threshold)/2$, but smaller than the threshold. In the third case, if the elapsed time between the last stored frame and the incoming frame is larger or equal to the storing interval, the incoming frame will be sent to the object detector and being stored. Otherwise, the incoming frame is ignored.

C. Feature-Aware Storage Allocator

As one of the primary goals of the proposed RTFI system is to enable efficient video clip retrieval with specified features, the feature-aware storage allocator is utilized to lower the retrieval latency by storing video frames with similar features on adjacent storage space. This is because, on hard disk drives, the disk head movement greatly denominates the read latency when retrieving stored data content. In other words, the read latency can be greatly reduced if the data is stored on continuous storage space. With this observation, the included feature-aware storage allocator aims to store video frames of the same object category in continuous storage space, known as *bucket*. The concept of the included space allocator is illustrated in Figure 4.

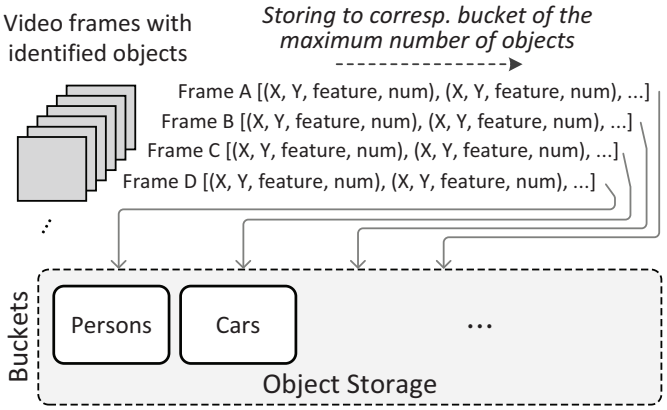


Fig. 4. The feature-aware storage allocator, in which videos are stored as separate video frames in bucket of the same object categories.

As shown in Figure 4, the storage space of the utilized object storage is divided into multiple buckets, and each bucket corresponds to one feature. Then, the object category with the highest number of objects in each video frame is identified, and the video frames are stored in the corresponding bucket of the highest number of objects. For instance, if a video frame

consists of 3 persons and 5 cars, this video frame is stored in the bucket of the car feature. On the other hand, the size of each bucket can be adjusted according to the size of each bucket by data migration. As video frames of the same feature are stored in adjacent storage space, the retrieval latency can be lowered when indexing a large number of video frames through specified features.

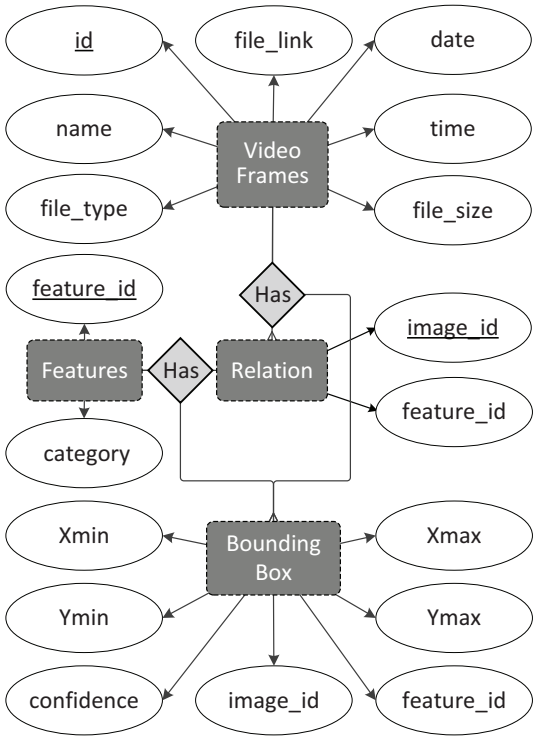


Fig. 5. The entity relationship (ER) diagram, which is utilized to facilitate the retrieval of video content that consists of certain features (*i.e.*, object category).

D. Feature-As-Metadata Indexing Scheme

To record the identified objects of each video frame and index video frames through features, the proposed RTFI system manages those stored video frames via their features in the MySQL-based Vitess database. First, the video frame table is created to record the basic information, including its identification number, file name, timestamp, file size, and the link to the stored frame in the object storage. Then, the bounding box table is created to track objects in video frames via bounding boxes, which is recorded in the table through the coordinates in video frames, confidence value, and feature labels. The confidence value refers to the probability that a bounding box actually contains an object. Next, the relation table is created to list all the detected features, such as cars and persons. On the other hand, the supported features of the utilized object detector are recorded in the features table. In the end, the utilized feature-as-metadata indexing scheme facilitates the management of stored video frames from the perspective of features and enables users to query video clips by specified features, timestamps, and locations. The above structure can be summarized as an entity-relationship (ER) diagram in Figure 5.

With the proposed scheme, for example, users can efficiently search for images based on features listed in the features table, acquire the bounding box information and image id from the bounding box table, and retrieve the video frames through the file link in the video frames table. Afterward, the video frames with the specified features can be retrieved from the object storage with the file link accordingly.

IV. Performance Evaluation

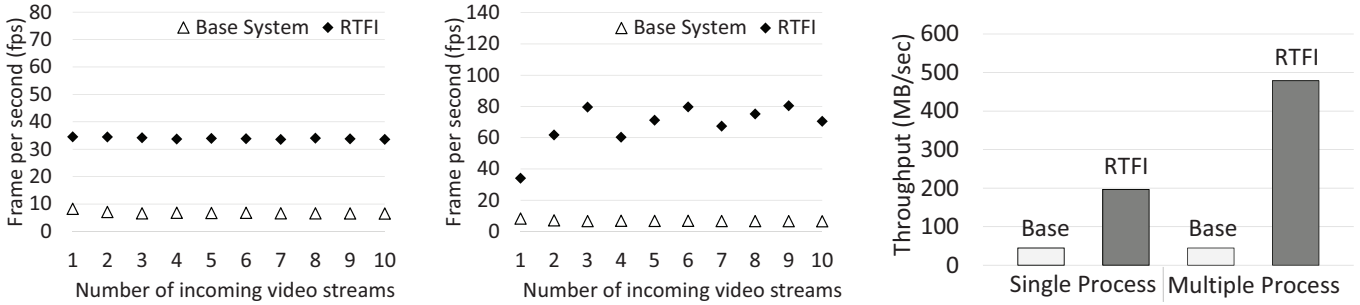
A. Experiment Setup

Experiments are conducted to evaluate the effectiveness of the proposed RTFI system regarding the number of frames that can be processed and stored per second, the system throughput, the storage space usage, and the number of identified objects. In addition, because similarity checks are performed within the proposed system to avoid unnecessary processing and lower storage space usage, the suitable threshold of the similarity check is also studied by considering the number of identified objects and the latency of the proposed RTFI.

The datasets utilized within this evaluation include public video datasets released by the Multiple Object Tracking Benchmark [2] and self-recorded videos consisting of moving peoples, cars, and stationary scenes. Those chosen videos from the public dataset are composed of a large number of constantly moving persons and cars. Self-recorded videos can be categorized into two groups. The videos of the first group are recorded on the streets with constantly moving cars and persons, while videos in the second group are mostly stationary scenes with little moving objects. Note that the self-recorded videos are recorded at 1080p with 60 FPS and will be made available after the acceptance of this paper. These datasets are then utilized to evaluate the implemented RTFI system on a Linux-based system, whose specification be summarized as Table II. Finally, the evaluation results of the RTFI system are compared with the base system consisting of Minio object storage and Vitess database without the proposed design. In the base system, the similarity deduplication module and feature-aware storage allocator are not enabled. The base system utilizes a single bucket to store all the capture videos and follows the proposed feature-as-metadata indexing for feature-based indexing.

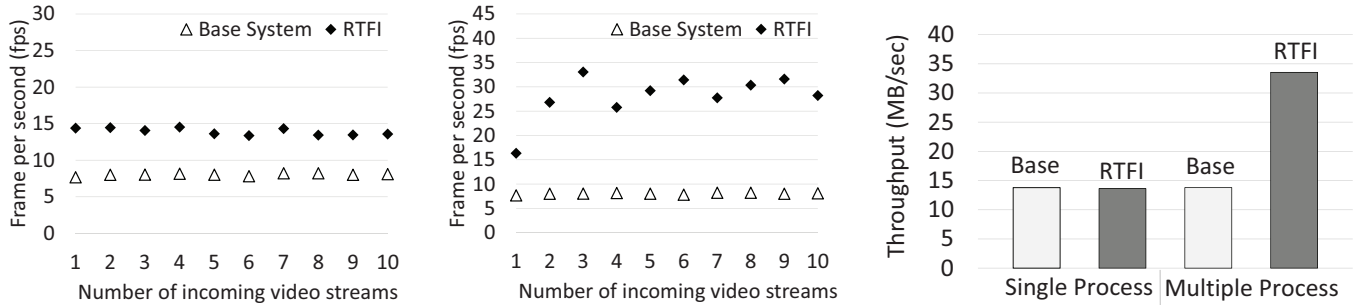
B. Experimental Results

In our implementation, both the single and multiple processes approaches are utilized to evaluate the proposed system and be compared with the base system. The FPS comparison of single and multiple processes implementations are summarized



(a) FPS with single process. (b) FPS with multiple process. (c) Throughput comparison

Fig. 6. Self-recorded dataset: Experimental results of frame per second (FPS) and throughput. The results show that the proposed RTFI can effectively improve the system throughput by an average of 10.70x. For single process implementation, the FPS is limited by the speed of object detection. Therefore, the FPS does not grow as the number of video streams increases. *For multiple process implementation, because 3 processes are utilized in this test, the FPS grows higher when the number of live video streams is the multiples of 3.*



(a) FPS with single process (b) FPS with multiple process (c) Throughput comparison

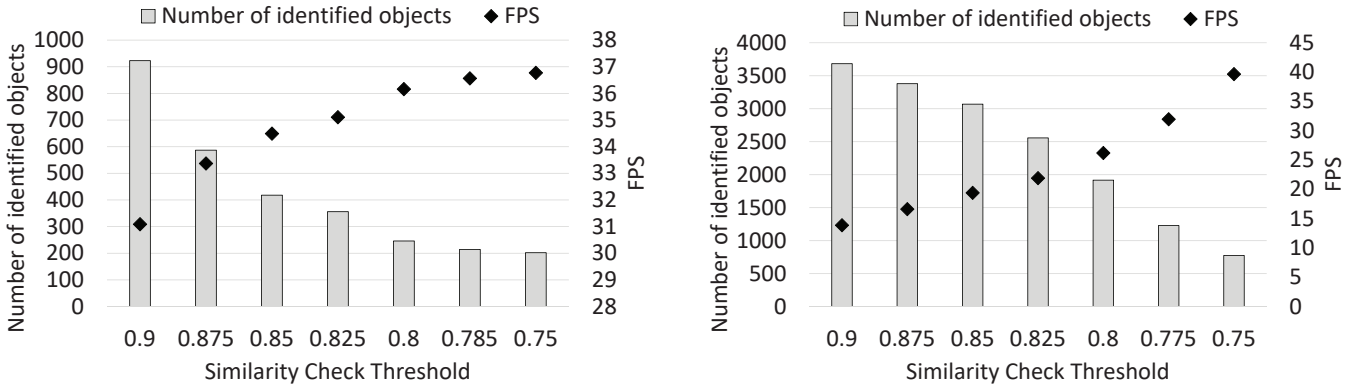
Fig. 7. Public dataset: Experimental results of frame per second (FPS) and throughput. The results show that the proposed RTFI can effectively improve FPS by 3.49x on average with multiple process implementation, even when the number of camera grows. In addition, the system throughput can also be improved by 2.43x.

TABLE II
SPECIFICATION OF THE EXPERIMENTAL COMPUTER.

Operating System	Linux kernel 4.15.0-52-generic
CPU	Intel CPU E5-2623 v3 @ 3.00GHz
GPU	Nvidia Titan X (Pascal) 12GB memory
RAM	32GB DDR4
Disk	Samsung SSD 850 EVO 1TB

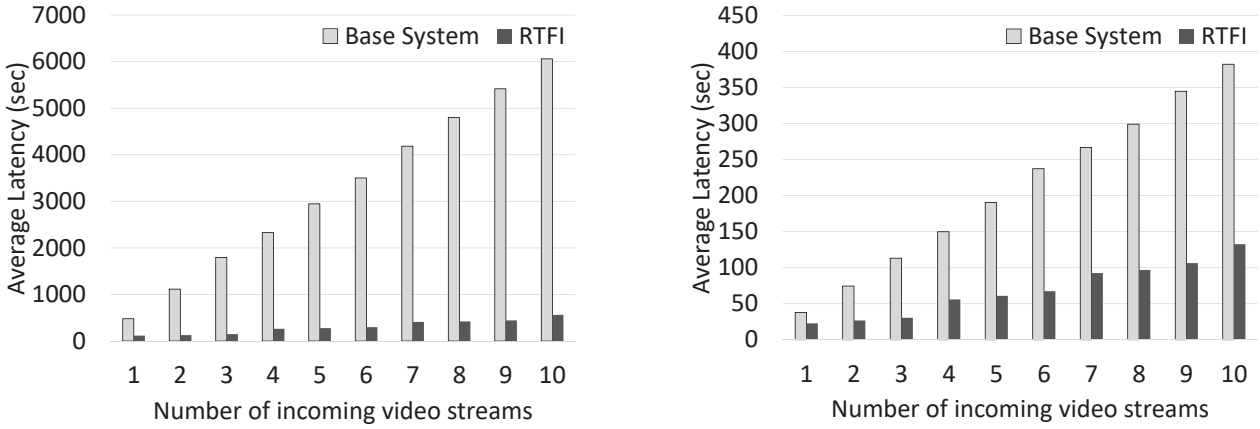
in Figures 6 and 7 for the self-recorded and public datasets, respectively. As shown in Figures 6(a) and 7(a), the proposed RTFI can effectively achieve an average of 33 and 13 FPS for the self-recorded and public video datasets with single process implementation. The performance difference between these two datasets can be attributed to the latency difference of object detection and storing because public video datasets have more objects per frame.

As shown in Figures 6(b) and 7(b), the proposed RTFI system can improve the number of processed frames per second by an average of 10.60x and 3.49x for the self-recorded and public datasets with the multiple processes implementation. In particular, the multi-process implementation can achieve up to 70 FPS for 10 live video streams and realize the goal of processing and storing video content with their identified object categories in real-time. In addition, the proposed system can also maintain almost identical FPS as the number of live video streams grows. Meanwhile, with the multi-process implementation, RTFI can achieve an average of 49 FPS for those investigated datasets. In other words, RTFI can make a video frame searchable with the identified object categories within 20 milliseconds. In the multi-process implementation, 3 processes are initialized to process and store the incoming live video streams. Notably, utilizing 3 processes also results in higher FPS when the number of live video streams is the



(a) Self-recorded dataset (b) Public dataset

Fig. 8. The comparison of the number of identified objects and latency with different similarity check thresholds. The average results suggests that 0.85 is a suit threshold for balancing between the latency and the number of identified objects.



(a) Self-recorded dataset (b) Public dataset

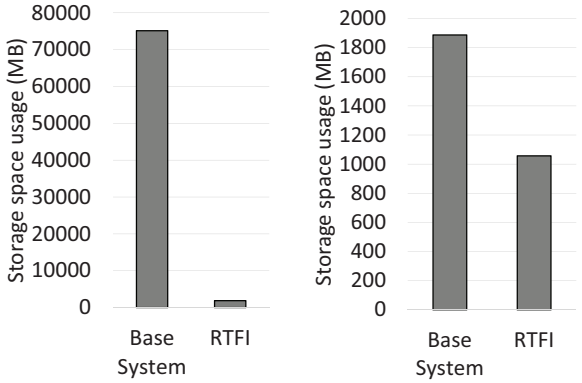
Fig. 9. Latency comparison with different numbers of video streams. The results show that the latency growth of the proposed RTFI is much smaller than that of the base system.

multiples of 3. From the above results, *it can be seen that directly utilizing the base system composed of Minio and Vitess without the proposed design can not effectively achieve the design goal of this study.*

In addition to the number of frames processed per second, the throughput of both the RTFI and base systems are also given under the single and multiple processes implementation. The throughput comparison is given in Figures 6(c) and 7(c) for the two utilized datasets. For the self-recorded dataset, the proposed RTFI system can effectively improve the throughput by 4.39x and 10.70x on average for single and multiple processes implementations. For the public dataset, the throughput improvements are 0.98x and 2.43x for single and multiple processes implementations. In summary, the average throughput improvement for all tested datasets is 6.56x on average with multiple processes implementations. Above results show that the proposed system achieves positive improvements in both the FPS and storage throughput with its included components.

Next, the threshold of the similarity check (see Section III-B) is also studied based on the input dataset of this evaluation. The important factor of deriving a suitable similarity threshold is striking a balance between the reduced latency for processing and storing videos and the number of missing objects. Then, we set the threshold between 0.9 and 0.75 to study the latency and the number of identified objects. According to the comparison results summarized in Figure 8, we determine the 0.85 is a suitable value for the similarity check. This is because, at the threshold of 0.85, the proposed RTFI system can effectively reduce the latency required for processing and storing videos, while avoiding identifying the same object across multiple frames that have very high similarity with each other. In other words, setting a suitable

threshold can also prevent the proposed RTFI from identifying and recording the same object multiple times across a series of similar frames.



(a) Self-recorded dataset (b) Public dataset

Fig. 10. The storage space usage comparison. The results show that the proposed RTFI can also reduce the storage space usage by 97.5% and 43.99% for the two utilized datasets.

Furthermore, the latencies of processing different numbers of video streams are reported. As shown in Figure 9, the results suggest that the proposed RTFI has smaller latency growth, when compared with the base system. With only one video stream, the latency reductions are 75.76% and 40/06% for the self-recorded and public datasets, respectively. With ten video steams, the latency reductions grow to 90.73% and 65.31%, respectively. Therefore, we can conclude that the proposed RTFI can effectively reduce the processing latency. Finally, storage space usage is also studied to investigate the reduced storage space usage of the proposed RTFI. As the comparison in Figure 10 suggests, the proposed RTFI can effectively reduce the storage space usage by 97.5% and 43.99% for self-recorded and public datasets, after deduplicating video frames that have high similarity to the previous video frames.

V. Conclusion

This study proposes a real-time feature indexing (RTFI) system for enabling video storing with the identified object categories and making the video content searchable in a real-time approach. RTFI aims at improving both the video process and retrieval performance by applying three different strategies - similarity deduplication module, feature-aware storage allocator and feature-as-metadata indexing scheme. Through extensive experimentation, we show that these strategies are effective in reducing the storage latency and the usage of disk space for feature-based indexing in a real-time fashion. In particular, the proposed system can boost the storing performance upto 10.60x, achieve the storing performance of upto 70 frames per second (FPS) with 3 processes implementation, and allow video content to become searchable in 20 milliseconds when there are 10 live video streams coming into the proposed system simultaneously. The future work of this study will be focused on extending the proposed system into distributed architecture for supporting a larger number of incoming video streams.

References

[1] Minio. https://min.io/.

[2] Multiple object tracking benchmark. https://motchallenge.net/vis/AVG-TownCentre.

[3] Vitess. https://vitess.io/.

[4] M. Babiker, O. O. Khalifa, K. K. Htike, A. Hassan, and M. Zaharadeen. Automated daily human activity recognition for video surveillance using neural network. In *2017 IEEE 4th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA)*, pages 1–5, Nov 2017.

[5] F. Bousetouane and B. Morris. Fast cnn surveillance pipeline for fine-grained vessel classification and detection in maritime scenarios. In *2016 13th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 242–248, Aug 2016.

[6] Z. Chen and X. Huang. End-to-end learning for lane keeping of self-driving cars. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1856–1860, June 2017.

[7] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98 – 115, 2015.

[8] Rahul Radhakrishnan Iyer, Sanjeel Parekh, Vikas Mohandoss, Anush Ramsurat, Bhiksha Raj, and Rita Singh. Content-based video indexing and retrieval using corr-lda, 2016. arXiv:1602.08581.

[9] S. Jha and P. Trivedi. An automated video surveillance system using viewpoint feature histogram and cuda-enabled gpus. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1812–1816, Aug 2013.

[10] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, February 2006.

[11] Seung-Hwan Lim, Steven Young, and Robert Patton. An analysis of image storage systems for scalable training of deep neural networks. 2016.

[12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. 2017. arXiv:1708.02002.

[13] Hao Luo, Wenxuan Xie, Xinggang Wang, and Wenjun Zeng. Detect or track: Towards cost-effective video object detection/tracking. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:8803–8810, 07 2019.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 2018. arXiv:1804.02767.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. arXiv:1506.01497.

[17] E. Salahat, H. Saleh, B. Mohammad, M. Al-Qutayri, A. Sluzek, and M. Ismail. Automated real-time video surveillance algorithms for soc implementation: A survey. In *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 82–83, Dec 2013.

[18] Martin Strohbach, Jörg Daubert, Herman Ravkin, and Mario Lischka. *Big Data Storage*, pages 119–141. 2016.

[19] Lei Tai and Ming Liu. Mobile robots exploration through cnn-based reinforcement learning. *Robotics and Biomimetics*, 3(1):24, Dec 2016.

[20] Y. Uchida, S. Sakazawa, and S. Satoh. Binary feature-based image retrieval with effective indexing and scoring. In *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, pages 319–320, Oct 2014.

[21] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.