This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391

CrossVision: Real-time On-Camera Video Analysis via Common RoI Load Balancing

Letian Zhang *Student Member, IEEE* Zhuo Lu *Senior Member, IEEE*
Linqi Song *Senior Member, IEEE* Jie Xu *Senior Member, IEEE*

Abstract—Smart cameras with on-device deep learning inference capabilities are enabling distributed video analytics at the data source without sending raw video data over the often unreliable and congested wireless network. However, how to unleash the full potential of the computing power of the camera network requires careful coordination among the distributed cameras, catering to the uneven workload distribution and the heterogeneous computing capabilities. This paper presents CrossVision, a distributed framework for real-time video analytics, that retains all video data on cameras while achieving low inference delay and high inference accuracy. The key idea behind CrossVision is that there is a significant information redundancy in the video content captured by cameras with overlapped Field-of-Views (FoVs), which can be exploited to reduce inference workload as well as improve inference accuracy between correlated cameras. CrossVision consists of three main components to realize its function: a Region-of-Interest (RoI) Matcher that discovers video content correlation based on a segmented FoV transformation scheme; a Workload Balancer that implements a randomized workload balancing strategy based on a bulk-queuing analysis, taking into account the cameras' predicted future workload arrivals; an Accuracy Guard that ensures that the inference accuracy is not sacrificed as redundant information is discarded. We evaluate CrossVision in a hardware-augmented simulator and on real-world cross-camera datasets, and the results show that CrossVision is able to significantly reduce inference delay while improving the inference accuracy compared to a variety of baseline approaches.

Index Terms—Distributed Deep Learning Systems, On-Device AI, Live Video Analytics, Workload Adaptive, Edge Computing.

1 Introduction

From road intersections to shopping malls and from university campuses to public squares, video cameras are ubiquitous nowadays to collect data for applications such as traffic control and security surveillance. Thanks to the recent breakthrough of deep learning (DL), we can now perform sophisticated video analytics tasks on the massive amount of data generated by these video cameras to retrieve key information with an unprecedented accuracy. Because DL-based video analytics is compute-intensive while traditional video cameras have extremely limited computing capability, video streams captured by traditional cameras have to be transmitted to a cloud or an edge server, which has sufficient computing power, to perform video analytics [1]–[5]. Although such an offloading-based approach is meaningful for legacy camera systems lacking suitable computing resources, it is not ideal as it relies heavily on the network and significantly stresses the network [6]. Not only the video analytics delay is highly sensitive to the network bandwidth between the cameras and the cloud/edge servers, but also it becomes infeasible to transfer all visual data over the network as the number of cameras scales up.

- *L. Zhang is with the Department of Computer Science, Middle Tennessee State University. Email: letian.zhang@mtsu.edu.*
- *J. Xu is with the Department of Electrical and Computer Engineering, University of Miami. Email: jiexu@miami.edu.*
- *Z. Lu is with the Department of Electrical Engineering, University of South Florida, Tampa, FL, 33620 USA. E-mail: zhuolu@usf.edu.*
- *L. Song is with the Department of Computer Science, City University of Hong Kong, Hong Kong, and also with the City University of Hong Kong Shenzhen Research Institute, Shenzhen 518057, China. Email: linqi.song@cityu.edu.hk.*

With the network being a bottleneck for real-time video analytics in large-scale camera networks, recent efforts have been on pushing video analytics directly onto the video cameras themselves [7]–[9], thereby mitigating the potential negative network impacts. Smart cameras, equipped with on-device DL accelerators, are increasingly being deployed and replacing traditional video cameras [10]. These smart cameras are not only able to perform basic video processing tasks such as background subtraction and motion detection, but also capable of executing complicated DL-based pipelines to detect and recognize the objects and a variety of their attributes. Therefore, smart cameras, while generating visual data, can also perform video analytics directly on this raw data without moving it over the network. However, although on-device video analytics eliminates the reliance on the network to a large extent, it has its own drawbacks due to the isolated data processing by individual cameras. First, depending on the time and what areas the cameras are covering, cameras are not equal in terms of the number of objects of interest and hence the workload of recognizing the captured objects and their attributes. Cameras in a hotspot (e.g., stadium entrance during a football game) can receive a large amount of workload that may even exceed their computing capability, resulting in significantly prolonged inference latency. Second, on-device video analytics performance is also limited by the quality of the cameras' individually captured data. Objects that are too far away and/or occluded can easily lead to wrong detection results and hence a reduced inference accuracy.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391

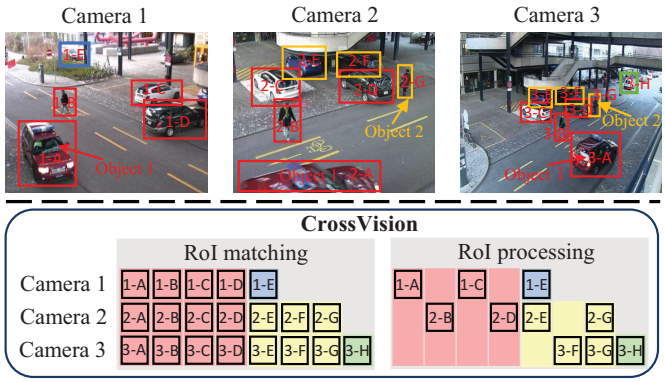


Fig. 1. A three-camera network with overlapped FoVs. CrossVision matches the RoIs of different cameras to the same objects, uses RoI redundancy to reduce and balance the system workload (e.g., the total workload reduces from 20 to 10), and improves the inference accuracy by fusing results of low quality RoIs (e.g., RoI 2-G and RoI 3-G).

1.1 CrossVision

In this paper, we present CrossVision, a distributed real-time cross-camera video analytics framework that overcomes the aforementioned drawbacks in densely deployed smart camera networks. CrossVision exploits the information redundancy across proximate smart cameras that have overlapped field-of-views (FoVs) to (1) adaptively balance workload among a network of smart cameras to reduce latency, and (2) intelligently fuse inference results from multiple smart cameras when necessary to improve accuracy. A salient feature of CrossVision is that *all visual data is retained at the smart camera that captures this data*; no visual data is transferred over the network to the cloud/edge server or other peer smart cameras. This has a significant advantage over other collaborative video analytics approaches that require raw data exchange among peer cameras.

As a motivating example, consider a three-camera network with overlapped FoVs illustrated in Fig. 1. The cameras have different numbers of objects of interests but some objects appear in the FoVs of multiple cameras. In particular, objects in red boxes appear in all three cameras, objects in yellow boxes appear in both Camera 2 and Camera 3 while the object in the green box appears only in Camera 3. Since the detection of Object 1 (i.e., the red car) by Camera 1 is enough to locate it on this road segment at this moment, Camera 2 and Camera 3 can save their computing resource for the inference of other objects. On the other hand, both Camera 2 and Camera 3 do not have a clear view of Object 2 (i.e., the women in black). Therefore, they can share their inference results with each other and fuse the results to improve the inference accuracy. Motivated by the above observation, CrossVision reduces both the total system workload and individual cameras' workload by assigning matched RoIs to designated camera for video analytics, meanwhile improving the inference accuracy by fusing the results of low quality RoIs.

1.2 Design Challenges and Our Contributions

CrossVision aims to minimize the inference latency and maximize the inference throughput of the camera network while improving the inference accuracy by harnessing the intrinsic correlation of video data across cameras with overlapped FoVs. We highlight three design challenges of CrossVision and our contributions:

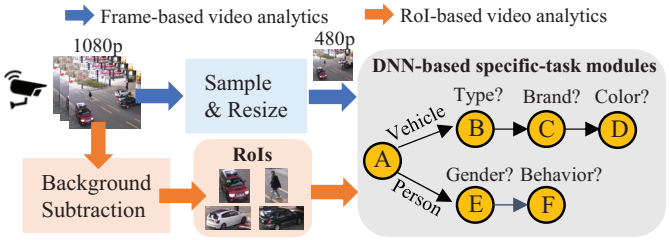


Fig. 2. Two types of video analytics pipelines.

Challenge 1: How to discover and describe the intrinsic video content correlation across the cameras? The first step is to identify common RoIs that appear in multiple cameras' FoVs. Unlike existing methods [11]–[13] that utilize deep image features from raw video frames, we develop a segmented FoV transformation method that establishes a pixel-to-pixel mapping between camera FoVs based on homography estimation [14]. RoIs are considered the same object if they occupy the same pixel positions after the transformation. The FoV transformation is applied to segmented FoVs to minimize distortion. With the identified common objects, we introduce "Virtual Camera", a concept that links smart cameras together via the common RoIs in their FoVs.

Challenge 2: How to balance the workload across cameras with heterogeneous on-device computing capacities and unpredictable workload dynamics? An object appearing in the FoVs of multiple cameras at the same moment may need to be inferred only once to save the computing resource and reduce the workload of cameras. Therefore, balancing workload across cameras is the key function of CrossVision in order to reduce the inference latency. To this end, we analyze the approximate inference latency based on the bulk queuing theory, and use the analysis result to formulate a workload optimization problem and design a randomized workload balancing algorithm while taking into account the predicted future workload.

Challenge 3: How to leverage common RoIs to improve inference accuracy? The improvement in inference latency is achieved by removing RoI redundancy via workload balancing. However, we also proactively preserve the redundancy when necessary in order to maintain a high level of inference accuracy. For example, if the same RoI in all FoVs has a low quality (e.g., small size), then CrossVision will ask multiple cameras to perform inference and fuse the results to improve the overall inference accuracy.

We build a hardware-augmented simulator to evaluate CrossVision. In this simulator, real-world cross-camera videos are processed by real hardware devices such as Nvidia Jetson TX2 and Nvidia Jetson Xavier, and we simulate the wireless network environment to investigate the network impact. Simulation results show that CrossVision is able to significantly reduce inference latency and improve throughput while achieving a high inference accuracy.

2 Background and Related Work

2.1 RoI-based Video Analytics Pipeline

Video analytics pipelines typically adopt a cascaded architecture consisting of many specific task modules to

perform various analytics. Besides identifying objects of interests, they can also perform task-specific analytics for each detected object of interest, e.g., determining the gender/behavior of a person, or the color/brand/type of a vehicle. As shown in Fig. 2, two pipelines are commonly used in existing live video analytics systems [15]. The first type is frame-based: raw video frames are first pre-processed and then fed directly into a cascading group of several pre-trained task-specific Deep Neural Network (DNN) models. The second type is RoI-based: a light-weight background subtraction method is used to extract the RoIs from the frame, and only these RoIs are fed into the cascading DNN-based analytics modules to obtain the results. Compared with the frame-based pipeline that has to constantly extract features from frames and perform inference, the RoI-based pipeline produces RoIs and performs inference on the RoIs only when RoIs appear in the frame, thereby reducing the computing usage on camera devices. For this reason, we focus on RoI-based video analytics in this work.

2.2 Related Work

Recognizing the limited computing capability of traditional camera devices, existing works [1]–[5], [16]–[20] frequently resort to cloud or edge servers for performing complicated DNN-based video analytics in camera networks. For example, in Couper [16], video data in a multi-camera network is sent to an edge computing cluster that deploys sliced DNN components. ANS [17] and JALAD [18] investigate adaptive collaborative inference between a camera device and an edge server. VideoEdge [19] dynamically configures the cameras and edge clusters to make tradeoff between resource usage and inference accuracy. However, video analytics in these works is performed without considering the intrinsic physical correlation among cameras. Maxim [20] proposes a learning-based framework to address the configuration adaptation problem in video analytics and employs a configuration sharing collaboration based on spatial and temporal correlations among cameras. Vigil [21] and CrossRoI [22] made similar observations as our paper that the same object may appear in multiple proximate cameras with overlapped FoVs. Vigil uploads only the frames that best capture the scene to the edge server for user's query when multiple cameras looking at the same scene capture different views of an object of interest. CrossRoI proposes to remove the repetitive appearances of the same RoI in multiple cameras before sending the frames to the edge server, thereby reducing the communication and computation cost. However, there are three main different aspects between CrossVision and all the aforementioned works. First, all the aforementioned works require offloading a significant amount of video data over a wireless network to the cloud/edge server, and hence the performance is heavily dependent on the network condition. Second, all the aforementioned works do not consider the workload balance in the video analytics, which may cause one or several cameras to be overloaded. Third, all the aforementioned works do not leverage the common RoIs across all the cameras to improve the overall inference accuracy.

As the cost and power consumption of on-device DL accelerators keep decreasing, fully on-device video analytics

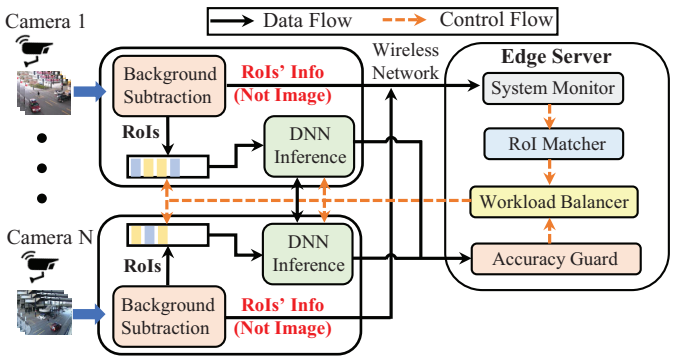


Fig. 3. System architecture of CrossVision. Note that all the video analytics are processed on the smart cameras, and the job of the edge server is mainly to control and coordinate the collaborative video analytics among the cameras by the RoIs' information of bounding box position (i.e., four pixel coordinates).

by smart cameras are becoming a reality. In this regard, Distream [23] also aims to balance the inference workload among distributed cameras. However, Distream ignores the video content correlation due to the overlapped FoVs of proximate cameras. As a result, workload balancing in Distream is achieved by migrating raw RoI images across cameras. Therefore, it still puts a heavy burden on the local wireless network. In addition, Distream misses the opportunity to improve the inference accuracy by fusing inference results of the same object. Physically migrating workload over wireless was also studied in other generic workload balancing problems [24]–[28]. Although these works are able to reshape the workload distribution, the total system workload remains the same and the wireless network can be further stressed due to the additional data transfer.

CrossVision establishes associations between the overlapping FoV of different cameras and matches RoI from different camera perspectives to balance workload and improve accuracy. Re-identification (ReID) algorithms [11]–[13] are commonly used in computer vision for object tracking problems to identify RoIs as the same object from different video frames. These algorithms use object detectors, such as deep neural network models, for object detection and extract deep image features from detected RoIs. The similarity between two detected RoIs is then computed based on their feature distance. However, in the case of CrossVision, the RoI matching only deals with the RoI bounding box position, i.e., four pixel coordinates, without touching raw video frames to avoid large transmission delay. Therefore, we propose an FoV transformation between two FoVs and use the pixel positions of RoIs to determine whether they correspond to the same object.

3 CrossVision Design

In this section, we present the design of CrossVision. We start with the coherent design for the overall architecture and then delve into the design details of four components – System Monitor, RoI Matcher, Workload Balancer and Accuracy Guard. To make it clear, the key notations used in this paper are summarized in Table 1.

3.1 Overall Architecture

As shown in Fig. 3, CrossVision is a distributed cross-camera video analytics framework that spans across a set of smart cameras and an edge server. We highlight that all the video analytics are performed on the smart cameras, and the job of the edge server is mainly to control and coordinate the collaborative video analytics among the cameras without touching the actual video data.

Data Plane. Each smart camera in the system captures video frames and performs background extraction to extract RoIs within each frame. The extracted RoIs are then stored in a local memory queue while the position information of the RoIs is transmitted to the edge server. Note that the position information of an RoI contains only the four-pixel coordinates of the bounding box vertices, and hence sending it to the edge server incurs a very low transmission cost. The edge server utilizes the received position information from all cameras to determine the optimal processing strategy for the RoIs. Control messages are sent back to the respective cameras, guiding them on whether to process the RoIs using their on-device inference engines or to mark them as overlapped RoIs that do not require local processing. Once processed, the inference results are reported back to the edge server.

Control Plane. Within the edge server, the System Monitor component continuously collects the position information of the reported RoIs from the cameras. This information is then forwarded to the RoI Matcher, which discovers correlations between RoIs in FoVs of different cameras, identifying whether they correspond to the same object. Based on these discovered correlations, the Workload Balancer assigns the inference tasks of the same object (i.e., RoI) to a specific camera for inference processing. The Workload Balancer periodically updates its workload balancing strategy by considering the predicted arrival rates of RoIs in the future. To ensure high inference accuracy, the Accuracy Guard verifies that the assigned camera possesses a sufficiently high-quality RoI (e.g., with a sufficient size), so as not to compromise accuracy. In scenarios where all cameras have low-quality RoIs of the same object, the Accuracy Guard instructs all cameras to perform on-device inference. After the inference results are reported back to the edge server, the Accuracy Guard aggregates the results to generate the final inference output.

3.2 System Monitor

System Monitor collects the RoIs' information (not raw images) from each smart cameras. The clocks of smart cameras and edge server are software synchronized and each camera has the same frame rate f. The RoI's information consists of the camera ID, coordinates of bounding boxes and timestamp, in the form of (*cameraID, left, top, width, height, timestamp*). *left* and *top* are the pixel coordinates locating the top left corner of the RoI bounding box, while the *width* and *height* information characterizes the bounding box size. *timestamp* represents the capture time of the corresponding RoIs. In this manner, the RoIs from smart cameras with the same indices are just image captures of the same scene at the same time from different perspectives. The System Monitor considers two cameras' timestamps as the same if their difference is small enough for frame alignment. i.e., $< \frac{1}{2f}$. *cameraID* is usually formed as the IP address, which will be used to send the workload balancing control message back to the smart cameras. After collecting all the RoIs' information from same timestamp as a group, System Monitor sends this group information to RoI Matcher to find the correlation of RoIs.

3.3 RoI Matcher

The goal of RoI Matcher is to determine whether RoIs in the overlapped FoVs of different cameras are indeed the same object. This would be easy if the coordinates of the RoIs could be calculated in a common world coordinate system. For cameras with depth information, such as stereo camera, 3D camera and RGB-D camera, it is possible to calculate such a world coordinate system provided with the depth information and camera calibration parameters [29]–[31]. However, for the vast majority of ordinary 2D cameras, RoI matching is non-trivial. Our idea is to establish a FoV transformation between two FoVs and use the pixel positions of RoIs to determine whether they correspond to the same object. The FoV transformation is calculated fully offline, and we perform RoI matching online as RoIs' positions are reported by the cameras. We discuss the steps in more details below.

(Offline) Segmented FoV Transformation. The key of FoV transformation is to find a transformation matrix H_{mn} between the overlapped FoVs of any two cameras m and n. Let (u_m^j, v_m^j) denote a pixel in camera m and (u_n^j, v_n^j) be the corresponding pixel in camera n, then the transformation matrix H_{mn} between the two coordinate systems is as follows,

$$w_{mn}^j[u_n^j, v_n^j, 1]^T = H_{mn}[u_m^j, v_m^j, 1]^T$$

where w_{mn}^j is the scale information between the pixel coordination system and the world coordination system. According to homography estimation [14], we can obtain the transformation matrix H_{mn} by solving a least squares problem, using four or more anchor pixel coordinates in camera m and their corresponding pixel coordinates in camera n. However, our experiment shows that a single transformation matrix for the entire FoV of a camera can cause a considerable transformation error because the anchor points marking is not exact. To mitigate the impact of estimation error during FoV transformation, we manually segment the FoV into several sub-areas, sample corresponding pixel points in each sub-area and calculate a transformation matrix for each sub-area. Fig. 4 shows that segmented FoV transformation significantly reduces the transformation errors. We note that FoV transformation is performed only once and in an offline manner for cameras with static FoVs. With these transformation matrices, pixel-to-pixel conversions can be calculated by traversing all the pixels in one camera. We offline build a hash table for each camera to store the map of its pixel coordinates to the possible corresponding pixel coordinates that may appear in other cameras.

(Online) RoI Matching. With the offline computed transformation matrices, we can compare RoIs for any two cameras to determine if they correspond to the same object.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391

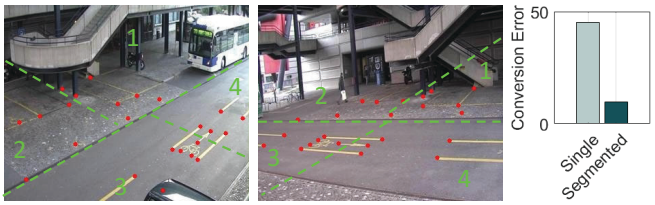


Fig. 4. Illustration of segmented FoV transformation.

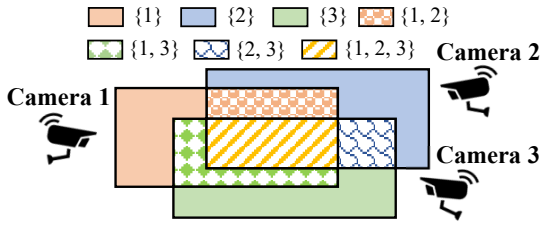


Fig. 5. Illustration of virtual camera. There are 7 virtual cameras: $\{1\}$, $\{2\}$, $\{3\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$.

However, even for the same object, its corresponding RoIs in different cameras can take different sizes and shapes, so it is nearly impossible to have a perfect matching between RoIs. Therefore, we consider two RoIs corresponding to the same object if the representative points (e.g., the center point, or a point closer to the ground) of the RoIs in two cameras are close enough (i.e., within a tolerance distance) after the coordinate transformation. However, it is still possible that two RoIs are indeed the same object but our method treats them as different objects. Hence, there is a tradeoff in the matching accuracy and efficiency by setting the tolerance distance. From a practical design perspective, one would prefer setting a small enough tolerance distance in order to have a correct matching with high confidence so that all objects are properly processed. Using the transformation matrices to match RoIs can still be a time-consuming process if the number of cameras is large and RoIs are many. To further reduce the computational time complexity, instead of actually calculating the transformation, RoI Matcher simply queries the hash tables built in the offline segmented FoV Transformation to determine if two RoIs match.

Example: We use RoIs in Fig. 1 as example. The RoI Matcher first uses the center pixel coordinates of RoI 1-A to query the Camera 1's hash table to get the possible corresponding pixel coordinates in Camera 2 and Camera 3. Based on these possible corresponding pixel coordinates, the RoI Matcher can find that the center pixel coordinates of RoI 2-A in Camera 2 and RoI 3-A in Camera 3 are close enough to the queried corresponding pixel coordinates from the Camera 1's hash table. Thus, the RoI matcher records that RoI 1-A in Camera 1, RoI 2-A in Camera 2 and RoI 3-A in Camera 3 are indeed the same object.

Virtual Camera. To better describe the correlations among the smart cameras and organize the matched RoIs, we introduce a concept called *Virtual Camera*. A virtual camera is a subset of physical cameras with overlapped FoVs, and we take the overlapped FoVs of these physical cameras as the FoV of the virtual camera. In terms of RoIs, if RoIs belonging to a set of physical cameras are successfully matched, then they also belong to the virtual camera representing this set of physical cameras. The number or the arrival rate of RoIs in virtual cameras thus reflects the correlation between physical cameras: the more RoIs a virtual camera sees, the stronger correlation the component physical cameras have. Fig. 5 illustrates the concept of virtual camera. As RoI Matcher performs RoI matching, it also counts the number of RoIs in every virtual camera. Note that matched RoIs are only counted once in a virtual camera.

We use $\mathcal{N} = \{1, ..., N\}$ to denote the set of physical cameras in the network and hence the set of virtual cameras, denoted by $\mathcal{V}$, is a power set of $\mathcal{N}$, i.e., $\mathcal{V} = 2^{\mathcal{N}}$. For example, cameras 1, 2 and 3 can form a virtual camera $v = \{1, 2, 3\}$ if they have overlapped FoVs. These notations will be useful when we describe the workload balancing strategy.

TABLE 1
Key Notations

Notations	Definitions
n	Camera index
t	Iteration index
T_{max}	Maximum number of iterations
F^a	Frame rate
X_n	Number of RoI inference tasks at camera n
λ_n	RoI inference task arrival rate at camera n
F_n^b	Batches per second processed by camera n
B_n	Batch size at camera n
μ_n	RoI processing rate at camera n
D_n	Average RoI inference delay at camera n
D_n	Upper-bound of D_n
v	Virtual camera index
λ_v	Predicted RoI inference task arrival rate at virtual camera v
$s_{v,n}$	Probability that workload of virtual camera v is assigned to physical camera n
$\tilde{\lambda}_n$	RoI inference task arrival rate at camera n after workload balance

Example: To further illustrate the virtual camera, we also use the RoIs in Fig. 1 as an example. After processed by the RoI Matcher, the matched RoIs in visutual camera are $R_{\{1\}} = \{\text{1-E}\}, R_{\{2\}} = \emptyset, R_{\{3\}} = \{\text{3-H}\}$, $R_{\{1,2\}} = \emptyset, R_{\{1,3\}} = \emptyset, R_{\{2,3\}} = \{\{\text{2-E, 3-E}\}, \{\text{2-F, 3-F}\}, \{\text{2-G, 3-G}\}\}, R_{\{1,2,3\}} = \{\{\text{1-A, 2-A, 3-A}\}, \{\text{1-B, 2-B, 3-B}\}, \{\text{1-C, 2-C, 3-C}\}, \{\text{1-D, 2-D, 3-D}\}\}$, and the number of RoIs in each virtual camera is $V_{\{1\}} = 1, V_{\{2\}} = 0, V_{\{3\}} = 1$, $V_{\{1,2\}} = 0, V_{\{1,3\}} = 0, V_{\{2,3\}} = 3, V_{\{1,2,3\}} = 4$.

3.4 Workload Balancer

Now, we are ready to introduce the core component of CrossVision – Workload Balancer. As its name suggests, Workload Balancer balances the workload in terms of RoI inference tasks across cameras by assigning the inference task of matched RoIs to a single smart camera. In this way, part of workload of cameras in hotspots can be migrated to cameras with light workload. There are a couple of key considerations for the design of Workload Balancer:

- *Heterogeneous Computing Capabilities*. Smart cameras are not all the same; their computing capabilities can vary depending on the hardware, the platform and the adopted DNN model. Hence, workload balancing needs to consider the cameras' heterogeneous computing capabilities to prevent any camera from becoming the bottleneck.
- *Future Workload Arrivals*. Workload in terms of RoI inference tasks varies across cameras and may change

over time. Workload balancing should not only look at the current workload of the cameras but also consider the possible impact of future incoming workload to avoid migrating workload to cameras that may experience heavy workload very soon.

Bulk Queue-based Inference Delay Analysis. Before we describe how to balance workload, we first perform an inference delay analysis to understand how the inference delay depends on the camera's workload.

Consider a representative camera n. Let F^a be the frame rate, and X_n be the random variable representing the number of RoI inference tasks in every frame. Note that without workload balancing, X_n would be the same as the number of RoIs seen in the frame. Thus, the RoI inference task arrival rate is $\lambda_n = \mathbb{E}(X_n)F^a$. DL-based video analytics typically processes tasks in batches, and hence the inference engine fetches multiple RoIs at a time from the RoI queue to process. Consider that camera n can process F_n^b batches per second with batch size being B_n. Thus, the RoI processing rate is $\mu_n = B_n F_n^b$. We are interested in characterizing the average RoI inference delay D_n, i.e., the time duration between when a RoI is extracted and when its inference result is obtained. This queuing system is not a simple M/M/1 queue, but features bulk arrivals (i.e., multiple tasks arrive at a time) and bulk processing (i.e., multiple tasks are processed at a time).

Proposition 1. The average RoI inference delay D_n is approximated by

$$D_n = \left[\frac{\rho_n\hat{\rho}_n}{1-\hat{\rho}_n} + \frac{\rho_n}{2}\frac{\mathbb{E}\left(X_n^2\right)}{\mathbb{E}\left(X_n\right)} + \frac{\rho_n}{2}\right]/\lambda_n + \frac{1}{\mu_n} \tag{1}$$

where $\rho_n = \frac{\lambda_n}{\mu_n}$, $\hat{\rho}_n = \exp(\gamma_n)$, $\gamma_n = \frac{2\mathbb{E}\left(X_n\right)(\rho_n - 1)}{\mathrm{Var}(\mathrm{X_n})\rho_n}$.

Proof. The proof is given in Appendix A. □

Since the RoI inference delay in Proposition 1 is quite complicated and hard to utilize, we further derive an upper bound on D_n, which has a much simpler form.

Proposition 2. Assuming $\frac{2\mathbb{E}\left(X_n\right)}{\mathrm{var}(X_n)} \geq 1 > \rho_n$, D_n can be upper-bounded by $\bar{D}_n$ as follows

$$D_n \leq \bar{D}_n \triangleq \frac{1}{\mu_n - \lambda_n} + \phi_n \tag{2}$$

where $\phi_n = \frac{3}{2\mu_n} + \frac{1}{2F^a}$.

Proof. The proof is given in Appendix B. □

Remark: For a standard M/M/1 queuing system [32] with arrival rate λ_n and service rate μ_n, its average waiting delay in system is

$$\tilde{D}_n = \frac{1}{\mu_n - \lambda_n} \tag{3}$$

which is essentially the first term in $\bar{D}_n$. Hence, the second term ϕ_n is the new addition due to batch arrival and batch processing, which reflects the extra impact of the computing capability on the inference delay. Nevertheless, workload balancing may also be performed based on the delay analysis using an M/M/1 approximation, albeit with a lower accuracy.

Computing Randomized Workload Balancing Strategy. To take into account the future workload arrivals, Workload Balancer periodically (e.g., every 10 minutes) computes a randomized workload balancing strategy using predicted future workload. Let λ_v be the predicted workload (i.e. RoI inference tasks) arrival rate for virtual camera v for the near future. This prediction can be made based on past workload arrival statistics, and various prediction methods, such as time-series-based [33] and neural network-based [34], have been developed. Since developing a new workload prediction algorithm is not the focus of this paper, we assume that $\lambda_v, \forall v \in \mathcal{V}$ is given to Workload Balancer, whose job is to distribute this workload among the physical cameras.

Let $s_{v,n} \in [0,1]$ be the probability that workload of virtual camera v is assigned to physical camera n. Clearly, for every $n \notin v$, $s_{v,n}$ must be 0, and $\sum_{n:n\in v} s_{v,n} = 1$ so that all workload is processed. Then the amount of workload that camera n will need to process is $\tilde{\lambda}_n = \sum_{v:n\in\mathcal{V}} s_{v,n}\lambda_v$. With this, we formulate an optimization problem as follows to find the optimal randomized strategy $\boldsymbol{s} = \{s_{v,n}; v \in \mathcal{V}, n \in \mathcal{N}\}$ that minimizes the worst-case inference delay.

$$\min_{\boldsymbol{s}} \quad \max_n \bar{D}_n \tag{4}$$

$$\text{s.t.} \quad \bar{D}_n = \frac{1}{\mu_n - \tilde{\lambda}_n} + \phi_n \tag{5}$$

$$\tilde{\lambda}_n = \sum_{v:n\in v} s_{v,n}\lambda_v, \forall n \in \mathcal{N} \tag{6}$$

$$\tilde{\lambda}_n < \mu_n, \forall n \in \mathcal{N} \tag{7}$$

$$\sum_{n:n\in v} s_{v,n} = 1, \forall v \in \mathcal{V} \tag{8}$$

$$s_{v,n} \in [0,1], \forall n \in v, \forall v \in \mathcal{V} \tag{9}$$

The above problem is a non-convex optimization problem due to the inference delay upper bound $\bar{D}_n$, which is generally hard to solve. Next, we develop an efficient algorithm to approximately solve this problem, inspired by the Coordinate Descent method [35]. Our algorithm consists of two stages. In the first stage, we solve a linear program to find an initial randomized strategy that satisfies all the constraints. In the second stage, we use a Coordinate Descent-inspired iterative procedure to improve the randomized strategy.

Stage 1: With $\bar{D}_n$ is monotonically increasing in $\tilde{\lambda}_n$. Therefore, instead of minimizing the maximum inference delay, we formulate a linear program to minimize the maximum workload assigned to the cameras as follows:

$$\min_{\boldsymbol{s}} \quad \max_n \tilde{\lambda}_n \tag{10}$$

$$\text{s.t.} \quad (6), (7), (8), (9) \tag{11}$$

Linear programs are easy to solve, and many commercial software packages exist to efficiently solve this problem. Let $\boldsymbol{s}_0$ be the optimal solution to (10). In fact, if the cameras have homogeneous computing capabilities, namely $\mu_n = \mu_m, \forall n, m \in \mathcal{N}$, then $\boldsymbol{s}_0$ is also the optimal solution to (4). If the cameras have heterogeneous computing capabilities, then $\boldsymbol{s}_0$ is a feasible solution to (4) because both problems have the same set of constraints. Note that if there does not exist a feasible solution to (10), then the overall workload is too large for the camera network to handle and hence, the system has to decrease the frame rate.

Stage 2: In the case of heterogeneous computing capabilities, we further use a Coordinate Descent-inspired iterative algorithm to improve on $\boldsymbol{s}_0$, which is shown in Algorithm 1. In every iteration t, we pick a pair of cameras i and j with overlapped FoVs to update their workload balancing strategies, i.e. $s_{v,i}, s_{v,j}, \forall v \in \mathcal{V}_{i,j}$, where $\mathcal{V}_{i,j}$ is the set of virtual cameras that contain both cameras i and j. Specifically, with probability $1-\epsilon$, we pick camera i as the one that has the largest inference delay, i.e., $i = \arg\max_n \bar{D}_n$ and randomly pick camera j among cameras that have overlapped FoVs with camera i. Because camera i currently has the worst inference delay performance, we update $s_{v,i}, \forall v \in \mathcal{V}_{i,j}$ in the negative gradient direction of $\bar{D}_i$ in an effort to reduce $\max_n \bar{D}_n$:

$$s_{v,i}^t \leftarrow s_{v,i}^{t-1} - \min\{\eta \nabla_{s_{v,i}} \bar{D}_i, s_{v,i}^{t-1}\}, \forall v \in \mathcal{V}_{i,j} \tag{12}$$

where $\eta > 0$ is the update step size. However, only updating the load balancing strategy of a single camera i violates the problem constraints and hence, we also update the strategy $s_{v,j}, \forall v \in \mathcal{V}_{i,j}$ to ensure that $s_{v,i} + s_{v,j}$ stays the same for all $v \in \mathcal{V}_{i,j}$. Therefore,

$$s_{v,j}^t \leftarrow s_{v,j}^{t-1} + \min\{\eta \nabla_{s_{v,i}} \bar{D}_i, s_{v,i}^{t-1}\}, \forall v \in \mathcal{V}_{i,j} \tag{13}$$

Although the updated $s_{v,j}$ may cause $\bar{D}_j$ to increase, it is less critical because camera j is not the bottleneck camera.

With probability ϵ, we randomly pick both cameras i and j in an iteration to introduce some exploration randomness in order to escape from local optimum solutions. The algorithm keeps the best workload balancing strategies so far, and terminates after a pre-determined maximum number of iterations T_{max}.

Algorithm 1 Stage 2 Algorithm

Input: $\lambda_v, \forall v \in \mathcal{V}$, $\mu_n, \forall n \in \mathcal{N}$, and $\boldsymbol{s}_0$
while $t < T_{\max}$ **do**
  Pick i:
    With prob. ϵ, $i = \arg\max_n \bar{D}_n^{t-1}$
    With prob. $1-\epsilon$, i is a random camera
  Randomly pick j so that i and j have overlapped FoVs
  Update strategies for i and j:

$$s_{v,i}^t \leftarrow s_{v,i}^{t-1} - \min\{\eta \nabla_{s_{v,i}} \bar{D}_i, s_{v,i}^{t-1}\}, \forall v \in \mathcal{V}_{i,j}$$
$$s_{v,j}^t \leftarrow s_{v,j}^{t-1} + \min\{\eta \nabla_{s_{v,i}} \bar{D}_i, s_{v,i}^{t-1}\}, \forall v \in \mathcal{V}_{i,j}$$

  Calculate $\tilde{\lambda}_j$
  if $\tilde{\lambda}_j > \mu_j$ **then**
    $s_{v,i}^t \leftarrow s_{v,i}^{t-1}, \quad s_{v,j}^t \leftarrow s_{v,j}^{t-1}$
  Calculate $\bar{D}_n^t, \forall n$
  if $\max_n \bar{D}_n^t < D_{\max}^*$ **then**
    $\boldsymbol{s}^* \leftarrow \boldsymbol{s}^t$, $D_{\max}^* \leftarrow \max_n \bar{D}_n^t$
Return $\boldsymbol{s}^*$

Workload Balancing Strategy Implementation. With the computed randomized strategy $\boldsymbol{s}$, implementing workload balancing is relatively straightforward. For each RoI matched to a virtual camera v, Workload Balancer randomly assigns the RoI to a physical camera $n \in v$ with probability $s_{v,n}$, and informs the cameras the assignment outcome via the control message. As also mentioned in the overall architecture, only the assigned camera processes the RoI using its on-device inference engine and later reports the inference result to the edge server, while the other cameras do not process their RoIs to save their on-device computing resources.

Example: We still use the RoIs in Fig.1 as an example. The matched RoI $\{2\text{-C}, 3\text{-C}\}$ is in virtual camera $v = \{2, 3\}$. Assume the workload balancing strategy of virtual camera $v = \{2, 3\}$ is $s_{\{2,3\},2} = 0.8, s_{\{2,3\},3} = 0.2$, then the match RoI $\{2\text{-C}, 3\text{-C}\}$ is assigned to be processed in physical camera 2 with probability 0.8, otherwise to physical camera 3. If matched RoI $\{2\text{-C}, 3\text{-C}\}$ is assigned to the physical camera 2, camera 3 will put the RoI 3-C in the waiting list and wait for the inference result of RoI 2-C from camera 2.

3.5 Accuracy Guard

Even for the same object, its RoIs in the FoVs of different cameras may appear very differently due to the distance, the camera view angles and even the light conditions. While workload balancing saves computing resources by removing the RoI redundancies, the RoI diversity can actually be used to improve the inference accuracy. The job of Accuracy Guard is exploiting this diversity to maintain a high inference accuracy in CrossVision. Three specific strategies are proposed, with the first two being proactive and the third being remedial.

Pick Cameras with High-Quality RoIs. The RoIs for the same object in different cameras have different qualities in terms of the RoI size. Intuitively, larger RoIs usually result in higher inference accuracy. If the RoI is too small in a camera and if this camera is picked to process the RoI, then the inference accuracy for this object will be compromised. Therefore, when picking the camera using the randomized workload balancing strategy, we set a minimal RoI size to ensure that the RoI size of the picked camera is large enough. Specifically, consider an object appearing in virtual camera v, and the respective RoI in camera $n \in v$ has size θ_n. We modify the randomized strategy as follows

$$\tilde{s}_{v,n} = \frac{s_{v,n} \cdot 1\{\theta_n \geq \theta_{\min}\}}{\sum_{j \in v} s_{v,j} \cdot 1\{\theta_j \geq \theta_{\min}\}} \tag{14}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, $\theta_{\min}$ is the required minimal RoI size. In this way, only cameras with high-quality RoIs have a positive probability to be selected.

Join the Power of Cameras with Low-Quality RoIs. It is possible that no camera has a high-quality RoI if the object is far away from all cameras that can see it. Hence, all cameras will have low-quality RoIs for this object. In this case, instead of picking a single camera to perform inference, Accuracy Guard asks all cameras to process their own RoIs and later fuses the submitted inference results. Again, consider an object appearing in virtual camera v, and every camera $n \in v$ does not have a high-quality RoI of this object. Take classification problems for example, the inference outcome of camera n on its own RoI is a vector of classification confidences $\{c_{n,k}\}_{k=1}^K$, where K is the total number of classes. A simple fusion rule of these inference outcomes is by taking the average:

$$c_k = \sum_{n \in v} c_{n,k}/|v|, \forall k = 1, ..., K \tag{15}$$

Then the final result is the top-ranked class in the combined confidence vector, i.e. $\arg\max_k c_k$. Other fusion rules can also be applied depending on the specific applications.

Seek Help If Uncertain. The above two strategies are proactive as they are taken before the cameras actually perform the inference task. In the third strategy, we aim to remedy the inference results if they are not satisfactory after the inference job is done. Specifically, a selected camera n can generate low-confidence inference outcome even if its RoI has a large enough size. In the application of object classification, this could be that the confidence of the top-ranked class is not high enough, i.e., $\arg\max_k c_{n,k} \leq c_{\text{th}}$, where c_{th} is the a confidence threshold. In this case, Accuracy Guard will seek help from cameras that were originally not selected to perform a second-round inference, and combine the inference outcomes from both the first-round and the second-round to produce the final inference result. Note that a camera keeps the RoI in the memory for a certain time period even if it is not initially selected to process the RoI. Clearly, since a second-round inference incurs an extra inference latency, there is a tradeoff by setting the confidence threshold c_{th}.

4 Evaluations

In this section, we build a hardware-augmented simulator to evaluate CrossVision on three real-world cross-camera datasets.

4.1 Dataset

EPFL dataset [36]: This dataset contains images taken by 6 static surveillance cameras with overlapped FoVs covering an area of $22m \times 22m$. At the same time, every camera takes a photo of the scene from different angles. Totally, there are 242 frames for each camera, and we duplicate them in our experiments. The sequence is recorded at the EPFL university campus where there was a road with a bus stop, parking slots for cars and a pedestrian crossing. A total number of 1297 persons, 3553 cars and 56 buses were manually annotated with a bounding box around them.

AI City dataset [37]: AI City Challenge 2021 traffic video dataset published by NVIDIA consists of two types of scenes where the traffic cameras are deployed either along long streets or around a traffic intersection, in a northern American city. We choose the most challenging scene of type two to evaluate CrossVision, where 5 cameras are deployed around a traffic crossing with complicated cross-camera viewpoint overlapping. Totally, there are 1955 synchronized frames for each camera in this dataset. Due to the large distortion of the fifth camera' view, we only use the first four cameras as the AI City dataset.

SALSA dataset [38]: SALSA is recorded in a regular indoor space and the captured social event involves 18 participants and includes two parts of roughly equal duration. The first part consists of a poster presentation session, where four research studies are presented by graduate students. In the second part, all participants are allowed to freely interact over food and beverages during a cocktail party. We use the cocktail party to evaluate CrossVision which consists of four synchronized static RGB cameras with overlapped FoVs covering an indoor lobby area.

4.2 Hardware-Augmented Simulator

The simulator simulates several smart cameras and an edge server in a wireless network. We implement CrossVision on real hardware devices, and simulate the wireless network by WiFi connection. We use WonderShaper [39] to set the wireless transmission speed to emulate different network conditions. The default value of network bandwidth is set to 15 Mbps if not specified otherwise. Nvidia Jetson TX2 and Nvidia Jetson Xavier devices are used as cameras of heterogeneous computing capabilities, and a Dell desktop is used as the edge server. For EPFL dataset, three simulated cameras are based on Xavier while the other three are based on TX2. For AI City dataset, two simulated cameras are based on Xavier while the other two are based on TX2.

We feed the images of one camera as a video stream to one simulated camera, and hence there are totally six video streams in EPFL dataset and four video streams in AI City dataset. We use OpenCV on the simulated camera to read the captured image and extract RoIs. A foreground mask is first calculated by subtracting a background model from the current frame, and then blob detection is performed to extract RoIs. Note that this is a light-weighted RoI extraction method which takes 5.6ms on Xavier and 7.9ms on TX2. We discard RoIs that are smaller than 0.5% of the frame size as they are simply too small for meaningful analysis. The local inference engine on the cameras uses MobileNetV2 [40], a state-of-the-art object detection DNN for mobile devices, and set the batch size to be 8 for batch processing.

4.3 Baselines and Performance Metrics

CrossVision is compared with the following baselines:

- **Standalone (SA)**: Each camera performs inference tasks on the extracted RoIs using its own on-device inference engine. No workload balancing is performed.
- **Centralized via Offloading (CO)**: The extracted RoI images in every frame on all cameras are sent to the edge server via wireless. The edge server performs the inference tasks on all received RoIs. The inference batch size is set as 32 in this case.
- **Distream:** Distream is a live video analytics system proposed in [23]. Like CrossVision, Distream performs video analytics based on extracted RoIs and aims to utilize the computing resources of the entire camera network. Different from CrossVision, Distream partitions the DNN inference between the smart cameras and the edge server, and balances the workload across cameras by migrating the RoI images across cameras.
- **CrossRoI:** CrossRoI [22] divides each camera's view into 24 spatial RoI masks. The goal of CrossRoI is to optimize the least number of RoI masks across all cameras with constraints as any object at each timestamp having at least one appearance region included by the RoI masks. Then, all the CrossRoI cameras crop their videos and only stream the areas included by the RoI masks to the edge server.
- **Myopic Workload Balancing (MWB)**: Within the CrossVision framework, we also consider a workload

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391

balancing strategy that utilizes only the current expected RoI waiting time of the cameras. Specifically, when a virtual camera v receives a RoI, it assigns the inference task to the camera $n \in v$ that currently has the shortest expected RoI waiting time. This is a myopic method as it does not consider the future incoming workload.

The following performance metrics are considered:

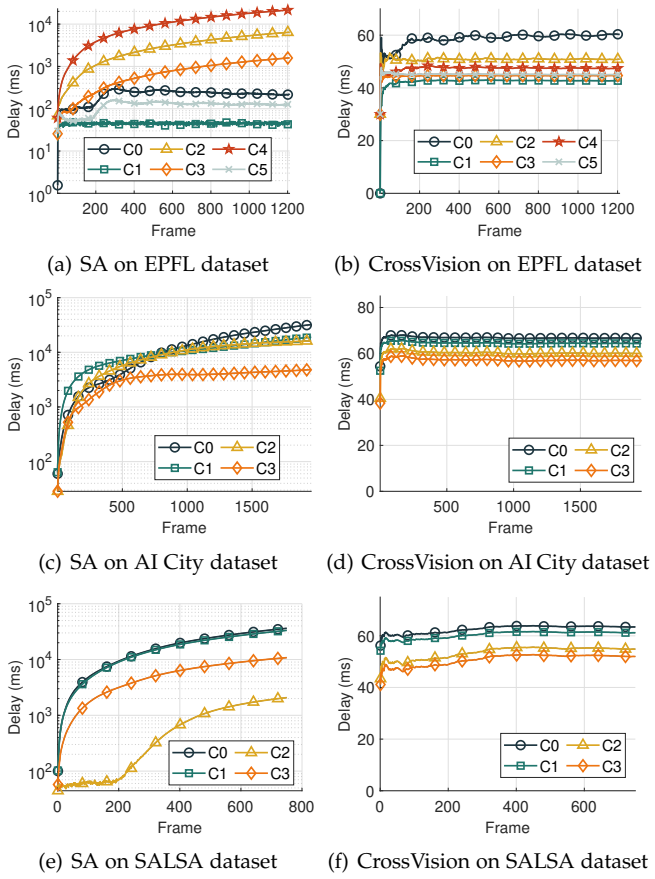
- **Frame Inference Delay**. Frame inference delay is defined as the elapsed time from when a frame is generated to when the analytics result of this frame is obtained (i.e., when all RoIs in this frame is processed). Note that the frame inference latency also includes the RoI extraction time, the network transmission time (due to data/control message exchange), the workload balancing time and the RoI queuing time in addition to the DNN inference time.
- **RoI Inference Throughput**. RoI Inference throughput is defined as the number of RoIs processed per second in the camera network.
- **Inference Accuracy**. For the considered object detection task, inference accuracy measures the percentage of RoIs that receive the correct detection result.

4.4 Evaluation Results

4.4.1 *Frame Inference Delay and RoI Inference Throughput*

We first show the advantage of workload balancing to reduce the frame inference delay by comparing CrossVision with SA on two datasets in Fig. 6. As we can see, the frame inference delay in SA on both two datasets (Fig. 6(a), Fig. 6(c) and Fig. 6(e)) increases to a very high, essentially unusable, value as frames are captured over time, i.e., 3 out of 6 cameras (Cameras 2, 3, and 4) in EPFL dataset and all 4 cameras in other two datasets. This is because the computing capability of these cameras can barely support their RoI inference task arrival rate. The RoIs spend a significant amount of time in the memory queue waiting to be processed, resulting in a very high overall inference delay. On the other hand, CrossVision exploits the overlapped FoVs between cameras to even the workload among the cameras according to their computing capabilities, thereby significantly reducing the frame inference latency. The highest frame inference delay is about 60ms among both datasets and hence, real-time video analytics is achieved. We also show the average workload on each camera in Fig. 7 on EPFL dataset, which further explains the above phenomenon. As can be seen, depending on the cameras' physical positions and view angles, the arrival workload is different, and the cameras have different computing capabilities. In standalone, the average workload generated at camera 2, 3, 4 are greater than their computing capability, which will eventually cause the infinite workload queuing time. However, CrossVision can significantly reduce the average workload by removing the RoI redundancy in the overlapped FoVs. This is very different from existing works (e.g., Distream), which migrates the RoI images across cameras.

In Table. 2 and Fig. 8, we further compare CrossVision with other baseline approaches that implement some sorts

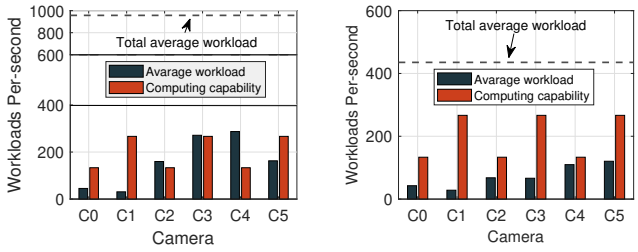


(a) SA on EPFL dataset (b) CrossVision on EPFL dataset

(c) SA on AI City dataset (d) CrossVision on AI City dataset

(e) SA on SALSA dataset (f) CrossVision on SALSA dataset

Fig. 6. Average frame inference delay of SA and CrossVision.



(a) SA (b) CrossVision

Fig. 7. Workload comparison of SA and CrossVision (EPFL dataset).

of workload balancing. The results confirm the superiority of CrossVision in terms of both frame inference delay and RoI inference throughput. Next, we explain why the baseline approaches are outperformed by CrossVision. **CO**: Although the edge server employed by CO is computationally much more powerful than any individual camera, it also has to process much more workload due to the centralized processing architecture. More importantly, all raw images have to be transmitted over the wireless network to the edge server, causing a significant communication time cost. **Distream**: Distream utilizes the distributed computing resources between individual cameras and edge server. Also, it re-shapes the workload distribution by sending RoI images from heavy-loaded cameras to light-load cameras. However, due to the transmission of RoI images over the wireless network, the communication time dominates the

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391

TABLE 2
Baseline comparison on frame inference delay.

	Frame inference delay (ms)					
EPFL	C0	C1	C2	C3	C4	C5
CO	387.1	331.6	559.1	878.9	365.5	542.6
Distream	123.8	127.5	121.8	128.7	112.9	116.8
CrossRoI	107.9	110.1	105.1	111.3	102.7	101.2
MWB	80.8	47.1	67.5	56.7	71.9	51.5
CrossVision	59.2	42.5	51.3	45.0	47.3	45.1

	Frame inference delay (ms)			
AI City	C0	C1	C2	C3
CO	962.5	424.1	527.0	624.4
Distream	206.1	163.1	175.6	199.7
CrossRoI	179.7	140.9	151.6	172.6
MWB	80.8	87.5	81.92	85.51
CrossVision	66.6	64.3	59.9	56.8

	Frame inference delay (ms)			
SALSA	C0	C1	C2	C3
CO	473.9	385.9	518.5	495.5
Distream	151.9	198.1	241.7	289.5
CrossRoI	132.4	171.1	208.6	250.2
MWB	86.5	77.71	72.1	69.26
CrossVision	63.4	61.1	54.8	51.9

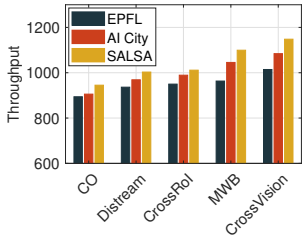


Fig. 8. Baselines comparison on RoI inference throughput.

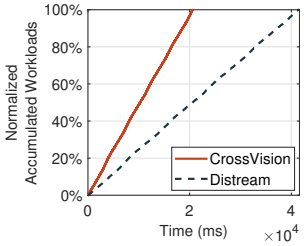


Fig. 9. Accumulated workload processing rate of CrossVision and Distream (EPFL dataset).

TABLE 3
Inference accuracy of SA, CrossRoI, CrossVision and CrossVision without accuracy guard (w/o AG).

	Inference Accuracy (%)					
EPFL	C0	C1	C2	C3	C4	C5
SA	67.7	62.1	94.6	73.3	71.7	92.3
CrossRoI	68.9	59.0	91.6	77.4	71.2	91.9
w/o AG	66.8	57.9	93.4	74.1	74.7	93.3
CrossVision	73.2	68.8	95.3	77.9	79.2	95.2

	Inference Accuracy (%)			
AI City	C0	C1	C2	C3
SA	94.6	93.3	91.7	92.7
CrossRoI	92.7	92.5	91.3	91.5
w/o AG	93.4	91.1	90.7	93.3
CrossVision	95.3	96.9	96.2	95.2

	Inference Accuracy (%)			
SALSA	C0	C1	C2	C3
SA	85.4	82.8	81.1	88.0
CrossRoI	85.2	88.9	79.5	91.3
w/o AG	87.3	90.1	79.3	88.9
CrossVision	95.7	97.7	94.3	96.5

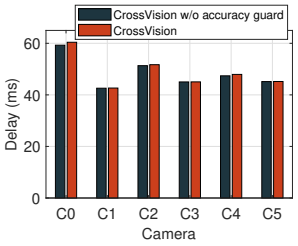


Fig. 10. Frame inference delay of CrossVision and CrossVision without accuracy guard.

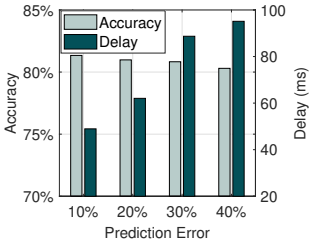


Fig. 11. Impact of workload prediction on the performance of CrossVision.

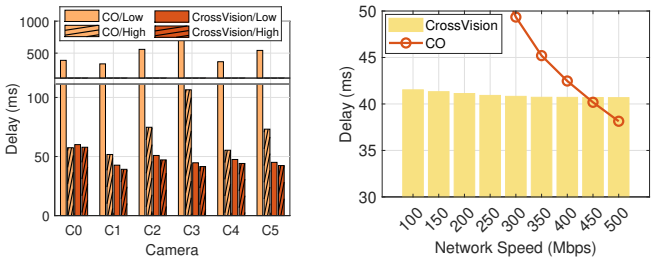
inference delay, especially when the network condition is poor. In addition, Distream ignores the redundancies in RoIs and processes all RoIs even if they may represent the same object. **CrossRoI**: Although CrossRoI harnesses the videos content association and redundancy across cameras' views to reduce the communication and computation costs, the network transmission latency of all the RoI masked images dominates the processing time. **MWB**: MWB adopts the CrossVision framework except that it myopically balances the workload using the current RoI expected waiting time. This strategy would be optimal if future workload could be freely dispatched to any camera in the network. However, if a camera is assigned with much workload now simply because of its currently short RoI expected waiting time, we may encounter an undesirable situation if the future workload is exclusive to this camera.

In Fig. 9, we further show the percentage of workload processed v.s. time for CrossVision and Distream. It is clear that CrossVision processes workload at a much faster rate than Distream. As mentioned above, this is mainly due to the fact that CrossVision removes many repetitive RoIs while Distream blindly processes all RoIs.

4.4.2 *Inference Accuracy*

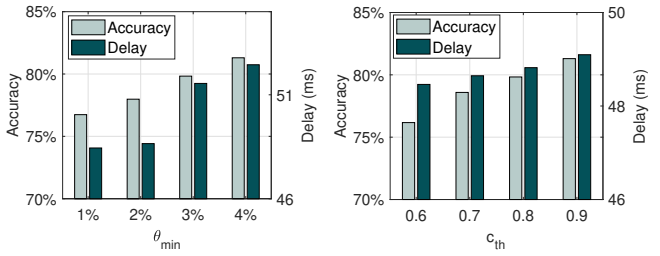
Table 3 presents a comparative analysis of the accuracy achieved by SA, CrossRoI, CrossVision, and CrossVision without Accuracy Guard. Because cameras have different viewing angles at and different distances from the object, the inference accuracy also varies across cameras for the same object. By performing workload balancing alone, the inference accuracy can be improved if the assigned camera has a high-quality RoI but can also be degraded if the assigned camera has a low-quality RoI. Overall, the average inference accuracy of CrossVision without Accuracy Guard is slightly lower than that of SA, with some individual cameras' inference accuracy being higher and others' being lower. With Accuracy Guard, CrossVision significantly improves the inference accuracy compared to SA, with an average improvement of 7.66% and up to 18.00% for individual cameras. Table 3 also reveals the inference accuracy achieved by CrossRoI and CrossVision. While CrossRoI employs similar RoI redundancy measures to enhance inference delay, it fails to leverage the opportunity to improve inference accuracy by fusing results from the same object. CrossVision, on the other hand, can achieve up to 18.62% improvement in inference accuracy compared to CrossRoI. Of course, the improved accuracy comes with additional overhead due to the increased overall workload and extra delay to remedy the inference result. In Fig. 10, we show the frame inference delay of CrossVision with and without Accuracy Guard. As we can see, the added delay is very small and hence, CrossVision with Accuracy Guard is able to achieve real-time video analytics at a high inference accuracy.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3301391



(a) (b)

Fig. 12. Impact of network on CO and CrossVision.



(a) Minimal required RoI size (b) Confidence threshold

Fig. 13. Impact of hyper-parameters θ_{min} and c_{th} on the performance of CrossVision.

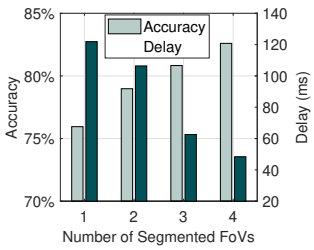


Fig. 14. Impact of segmented FoV transformation on the performance of CrossVision.

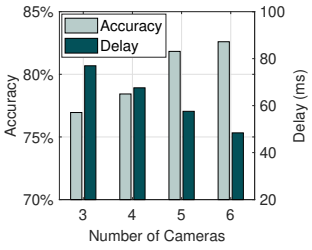


Fig. 15. Impact of number of overlapping cameras on the performance of CrossVision.

4.4.3 Impact of Workload Prediction

As aforementioned in Section 3.4, Workload Balancer periodically computes a randomized workload balancing strategy using predicted future workload. In this section, we set the workload prediction error from 10% to 40% and evaluate the impact of workload prediction on the performance of CrossVision. As we can see in Fig. 11, the average frame inference delay increases with the prediction error, because a larger prediction error causes the Workload Balancer to fail to find the optimal balancing strategies. However, the workload prediction has little impact on the inference accuracy since the Accuracy Guard exploits the RoI diversity to maintain a high inference accuracy.

4.4.4 Impact of Network

We now investigate the impact of the network transmission rate on the video analytics system. Fig. 12(a) reports the frame inference delay for CO and CrossVision in two representative network environments, namely a low rate at 10Mbps and a high rate at 100Mbps, which cover the typical bandwidth range in video surveillance systems on the market. As expected, the offloading-based approach CO is very sensitive to the transmission rate and gains a considerable improvement when the transmission rate increases. On the other hand, CrossVision is less sensitive to the network rate change as only a small amount of control information needs to be exchanged. Note that, even in the high rate case, CrossVision outperforms CO because CO has to process all RoIs by a single server while CrossVision only processes a subset of RoIs by removing the repetitive ones and utilizing the computing resources of all cameras. Fig. 12(b) separately shows the frame inference delay of Camera 3 for CO and CrossVision under network speed range from 100Mbps to 500Mbps. The results further prove that CrossVision is less sensitive to the network rate change. Note that CO outperforms CrossVision after 450Mbps, as the transmission delay of extracted RoI images is low enough for the powerful edge server to reap its advantages.

4.4.5 Impact of θ_{min} and c_{th}

The design of Accuracy Guard involves two system hyper-parameters: 1) θ_{min} minimal required RoI size; 2) c_{th} confidence threshold. We evaluate the impacts of these system hyper-parameters on the performance of CrossVision in Fig. 13. We can see that increasing θ_{min} and c_{th} leads to a higher inference accuracy, since CrossVision relies more on Accuracy Guard to improve the inference accuracy. On the other hand, the higher reliance causes larger delay due to the increased overall workloads.

4.4.6 Impact of Segmented FoV Transformation

The FoV transformation is calculated to find a transformation matrix between any two cameras. Rather than employing a single transformation matrix for the entire FoV of a camera, which may cause significant transformation errors due to imprecise anchor point marking, we manually segment the FoV into multiple sub-areas. We then sample corresponding pixel points in each sub-area and compute a transformation matrix for each sub-area. The impact of manual segmentation on the performance of CrossVision is depicted in Fig. 14. The result shows that increasing the number of segmented FoVs leads to higher inference accuracy and lower inference delay. This is because increased segmentation of FoVs reduces transformation errors, resulting in greater accuracy of RoI matching. Enhanced accuracy of RoI matching enables the system to better identify and associate objects seen by multiple cameras, which in turn allows for more cameras to be utilized in eliminating RoI redundancy and reducing inference delay. This increased camera coverage also provides more diverse perspectives of the object, which can improve inference accuracy by utilizing the diversity of RoIs from different camera angles.

4.4.7 Impact of Number of Overlapping Cameras

To illustrate the impact of the number of overlapping cameras, we selected various camera subsets from the EPFL dataset. As depicted in Fig. 15, the inference delay reduces as the number of cameras increases, while the inference accuracy improves with an increase in the number of cameras. This is due to the fact that more cameras with overlapping FoV can reduce the average workload by eliminating RoI

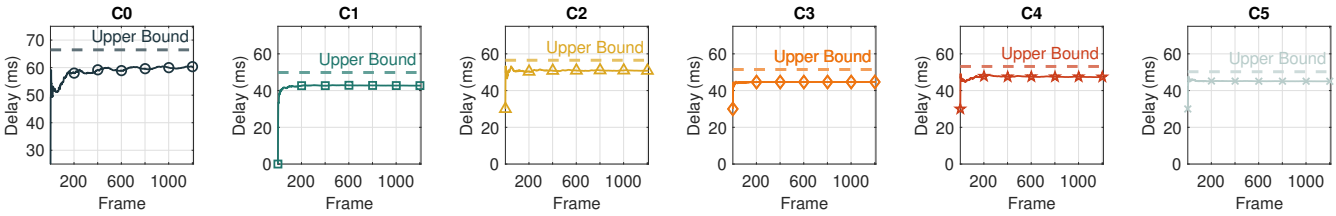


Fig. 16. Upper bound after workload balance.

TABLE 4
Component-wise analysis of CrossVision.

	Frame inference delay (ms)					
	C0	C1	C2	C3	C4	C5
CrossVision-L	80.9	48.8	66.7	52.9	66.7	53.9
CrossVision	59.2	42.5	51.3	45.0	47.3	45.1
Improvement	27%	13%	23%	15%	29%	16%

redundancy. Additionally, an increase in the number of perspective views of the object provides greater opportunities to improve inference accuracy through RoI diversity.

4.4.8 *Component-wise Analysis*

CrossVision performs cross-camera workload balancing considering the cameras' heterogeneous computing capabilities. Since the RoI inference delay in Proposition 1 is quite complicated and hard to utilize, the workload balance problem is to minimize the worst-case inference delay of upper bound on RoI inference delay. To solve the problem, CrossVision first calculates a feasible strategy assuming that all cameras have the same computing capabilities (i.e, stage 1 in section 3.4). Next, CrossVision uses a Coordinate Descent-inspired iterative algorithm to improve the above strategy by incorporating the heterogeneous computing capabilities (i.e, stage 2 in section 3.4). In this experiment, we first show how the proposed upper bound helps the Workload Balancer to find the optimal balancing strategy in Fig. 16. As we can see, by optimizing the worst-case inference delay of upper bound, CrossVision can balance the workloads across cameras. Then, we inspect the contributions of these two components by implementing **CrossVision-L**, which only uses the stage 1 strategy. Table 4 shows the comparison results on the frame inference delay. As can be seen, CrossVision improves the frame inference delay by $13\% - 29\%$ compared to CrossVision-L. This highlights the importance of stage 2 optimization in CrossVision and the necessity of considering cameras' heterogeneous computing capabilities.

5 Conclusions

Moving DL functionalities to edge devices, such as smart cameras in this paper, has been a recent trend in both the academia and the industry. CrossVision developed in this paper complements this trend by enabling collaborative DL-based video analytics in a cross-camera system, which not only reduces the inference latency but also improves the inference accuracy. This is achieved by recognizing and efficiently exploiting the physical correlation among proximate cameras with overlapped FoVs: inference latency is reduced by removing the RoI redundancy while inference accuracy is improved by utilizing the RoI diversity. Our design is both theoretically sound and practically effective. A limitation of the current framework is that the consideration of only static cameras. When cameras are moving (both heading and location), matching RoIs and balancing workload among the cameras are expected to be much more challenging. This will be explored in our future work.

References

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

[2] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 377–392.

[3] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[4] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–13.

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

[6] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[7] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 82–95.

[8] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[9] J. He, G. Baig, and L. Qiu, "Real-time deep video analytics on mobile devices," in *Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021, pp. 81–90.

[10] P. Natarajan, P. K. Atrey, and M. Kankanhalli, "Multi-camera coordination and control in surveillance systems: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 11, no. 4, pp. 1–30, 2015.

[11] Z. He, Y. Lei, S. Bai, and W. Wu, "Multi-camera vehicle tracking with powerful visual features and spatial-temporal cue." in *CVPR Workshops*, 2019, pp. 203–212.

[12] P. Li, G. Li, Z. Yan, Y. Li, M. Lu, P. Xu, Y. Gu, B. Bai, Y. Zhang, and D. Chuxing, "Spatio-temporal consistency and hierarchical matching for multi-target multi-camera vehicle tracking." in *CVPR Workshops*, 2019, pp. 222–230.

[13] E. Ristani and C. Tomasi, "Features for multi-target multi-camera tracking and re-identification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6036–6046.

[14] E. Dubrofsky, "Homography estimation," *Diplomová práce. Vancouver: Univerzita Britské Kolumbie*, vol. 5, 2009.

[15] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[16] K.-J. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: Dnn model slicing for visual analytics containers at the edge," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 179–194.

[17] L. Zhang, L. Chen, and J. Xu, "Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning," in *Proceedings of the Web Conference 2021*, 2021, pp. 3111–3123.

[18] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*. IEEE, 2018, pp. 671–678.

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

[20] R. Zhang, Y. Zhou, F. Wang, and Z. Wang, "Maxim: Drl-based cross-camera streaming configuration for real-time video analytics," in *2022 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2022, pp. 01–06.

[21] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015, pp. 426–438.

[22] H. Guo, S. Yao, Z. Yang, Q. Zhou, and K. Nahrstedt, "Crossroi: cross-camera region of interest optimization for efficient real time video analytics at scale," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 186–199.

[23] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 409–421.

[24] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[25] M. A. Islam, S. Ren, G. Quan, M. Z. Shakir, and A. V. Vasilakos, "Water-constrained geographic load balancing in data centers," *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 208–220, 2015.

[26] H. Xu, C. Feng, and B. Li, "Temperature aware workload managementin geo-distributed data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1743–1753, 2014.

[27] J. Luo, L. Rao, and X. Liu, "Spatio-temporal load balancing for energy cost optimization in distributed internet data centers," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 387–397, 2015.

[28] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening geographical load balancing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 39, no. 1, pp. 193–204, 2011.

[29] Q. Liu, T. Han, B. Kim *et al.*, "Livemap: Real-time dynamic map in automotive edge computing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021.

[30] M. Ding, Z. Wang, J. Sun, J. Shi, and P. Luo, "Camnet: Coarse-to-fine retrieval for camera re-localization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2871–2880.

[31] H.-N. Hu, Q.-Z. Cai, D. Wang, J. Lin, M. Sun, P. Krahenbuhl, T. Darrell, and F. Yu, "Joint monocular 3d vehicle detection and tracking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5390–5399.

[32] S. Zheng and A. F. Seila, "Some well-behaved estimators for the m/m/1 queue," *Operations Research Letters*, vol. 26, no. 5, pp. 231–235, 2000.

[33] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 1287–1294.

[34] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Neural network-based performance prediction for task migration on s-nuca many-cores," *IEEE Transactions on Computers*, 2020.

[35] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[36] G. Roig, X. Boix, H. B. Shitrit, and P. Fua, "Conditional random fields for multi-camera object detection," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 563–570.

[37] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, Y. Yao, L. Zheng, P. Chakraborty, C. E. Lopez, A. Sharma, Q. Feng, V. Ablavsky, and S. Sclaroff, "The 5th ai city challenge," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2021.

[38] X. Alameda-Pineda, J. Staiano, R. Subramanian, L. Batrinca, E. Ricci, B. Lepri, O. Lanz, and N. Sebe, "Salsa: A novel dataset for multimodal group behavior analysis," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 8, pp. 1707–1720, 2015.

[39] V. Mulhollon. (2004) Wondershaper. [Online]. Available: https://github.com/magnific0/wondershaper.

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[41] E. Parzen, *Stochastic processes*. SIAM, 1999.

[42] E. Gelenbe, "On approximate computer system models," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 261–269, 1975.

[43] S. Chiamsiri, *Diffusion Approximations for bulk queues and Inventory control models*. University of Missouri-Columbia, 1979.

[44] J. D. Little and S. C. Graves, "Little's law," in *Building intuition*. Springer, 2008, pp. 81–100.