Deploying Collaborative Machine Learning Systems in Edge with Multiple Cameras

Si Young Jang
KAIST
Daejeon, Korea

Utku Günay Acer
Nokia Bell Labs
Antwerp, Belgium

Chulhong Min and Fahim Kawsar
Nokia Bell Labs
Cambridge, UK

***Abstract*—Advancement in hardware capability has opened up the possibility of performing ML inference tasks at the edge using a large volume of sensory data generated from IoT devices such as cameras. As cameras become more pervasive, edge systems need to process streams from multiple sources with overlapping fields-of-view. In this position paper, we describe a collaborative sensing mechanism at the edge for such cases. We introduce a View Mapping Database (DB) that maps regions in a camera's field of view to regions in other cameras' view. We analyze characteristics of 5 video streams that capture an intersection from multiple angles, prototype a View Mapping DB, and present our preliminary results.**

I. Introduction

The recent rapid advancement in hardware of edge accelerators has led to the deployment of cloud scale machine learning (ML) inferences to run locally, near sensory data sources such as cameras and microphones, without the need for sending large volumes of data to remote data centers. This leads to improved efficiency, latency, and throughput.

Edge devices typically provide inference capabilities using already trained models. A typical ML inference task involves i) preparing the data acquired from a real-world sensor (e.g., pixels from an image sensor or waveforms from a microphone) to a compatible input format, ii) executing the model within a model-specific framework (e.g., TensorFlow or PyTorch) and iii) serving the inferences through well-defined interfaces.

While there are significant efforts to execute ML models in edge devices [1], [2], the usual approach to edge inference involves a single data source and a model. On the other hand, with sensors like cameras getting more and more pervasive, there is a growing need for deploying collaborative sensing mechanisms at the edge. Sensor fusion methods are developed to combine data from multiple sensors [3], [4]. However, they often lose the benefit of edge computing since they need to bring sensor data from multiple observers to a central server for model processing.

In this position paper, we describe a collaborative sensing mechanism for edge settings. Our mechanism is based on a modification of SensiX [5], namely SensiX++, which provides an end-to-end, multi-tenant platform to bring ML inference tasks on edge devices. We move some of the SensiX functionality to a central computing location, but unlike sensor fusion, it only processes locally executed model outputs from edge devices instead of raw sensor streams. For classification models, this sensor fusion is related to data aggregation techniques [6]. Our focus in this paper instead, is on the vision models and multi-camera video analytics that involve object/event detection and tracking across multiple cameras. While this is easier in cameras with non-overlapping fields of view (FOV), detection of the same object/event in multiple cameras needs to be taken into account for a more accurate representation.

We consider the case where each camera is connected to an edge-accelerator that is capable of running ML models based on neural networks. The model output, rather than the sensory data, is transferred to a local, edge-scale compute location, referred to as *edge cloud*. Edge cloud hosts multiple services to facilitate collaborative visual sensing in addition to interpreting the model output from multiple observers.

To support efficient multi-camera video analytics, it is important to distinguish identical elements, e.g., events and objects, in all cameras with overlapping FOV so that an ML task does not deal with redundant objects. A trivial approach to do this involves identifying each object in all camera frames by computing/retrieving the feature vectors of each object and evaluating the similarity between them, which requires heavy computation. Instead, we create a *View Map* that maps various regions across different cameras. To do this, we calibrate the cameras in an *offline* phase during which object identification is used to find anchor objects to obtain such mappings. Once the location of a target object is known in a camera view, in the form of a bounding box retrieved from the locally running detection model, we use a View Mapping DB to match it to objects retrieved from other cameras in the *online* phase. This view mapping is used in both spatial and temporal queries instead of identifying each object in each camera frame, thereby significantly reducing the compute and networking overhead.

This paper presents our on-going work on bringing inferences from multiple observers together to facilitate queries that require addressing the separation of cameras both spatially (location objects across different cameras) and temporally (tracking one mobile object across time). To this end, we first present the related work in Section II. We then describe our

Si Young Jang was an intern with Nokia Bell Labs when this work was conducted

978-4-907626-48-8/21/$31.00 ©2021 IPSJ

proposed system design in Section III. An early evaluation of some system components is given in Section IV. We conclude the paper in Section V.

II. Related work

Existing work on Video Analytics (VA) at the edge takes the advantage of computation power of edge or cloud to run real time VA pipeline which consists of multiple ML operations. However, the majority of VA systems are designed to perform VA pipeline on video streams individually [7], [8]. While DNN model compression & pruning [9], [10] and ML inference offloading [11], [12] & VA task placement optimization [13], [14] can be done per stream for optimization, still resource consumption grows linearly as the number of video feeds increases.

To resolve such issue, recent VA systems incorporate learning spatial and temporal relationships between the video feeds. Especially, these systems [15], [16] learns the spatial and temporal relationship between the video feeds obtained from geo-distributed locations where cameras' FOV does not overlap. The intuition here is that objects seen at a camera's (namely, camera A) FOV are likely going to be seen in other cameras' (namely, camera B or C) FOV after some time (e.g., 5 seconds between cameras A and B, 10 seconds between cameras A and C). Such spatial and temporal relationship between the cameras is leveraged to reduce the number of video frames to be analyzed, which leads to efficient resource usage.

In other systems [17]–[19], authors analyze video streams from geo-distributed cameras where the FOV overlaps. As multiple cameras are viewing the same area concurrently, analyzing all the frames results in poor utilization of resources. In [17], [19], authors propose a collaborative camera VA system which decides among all cameras, which camera's video frame should be transmitted: solely based on the maximum number of objects in each FOV [17], or based on the similarity of the object seen from two FOVs using reference points [19]. In CrossROI [18], the system learns the spatial relationship between *regions* of each camera's FOV and adjusts region-of-interest (ROI) masks of the FOV during runtime so to provide timely yet accurate VA while reducing network load. While a View Map DB in our system also learns spatial relationships between regions of different cameras, we use this information to identify common objects completely relying on the model output rather than sending some or all of the frames to a cloud server.

III. System Description

SensiX++ brings ML inference tasks to the edge devices. SensiX++ decomposes ML tasks into a number of components. A *data coordinator* component manages the sensors and collects raw data taking model requirements such as sampling rate, image resolution, etc. A *featurisation* container is dedicated to a number of pre-processing functions to prepare sensor input for model execution. These functions include model adaptation [20], translation [5], etc. This data is fed into a number of model containers, each equipped with necessary tools to execute a neural network with a CPU, GPU or another processing platform such as TPU.

SensiX++ allows the execution of functions that process the outcome from model containers. These functions may serve several purposes including generating higher order of analytics and annotations for the sensor data using predictions from one or more models or other functions. SensiX++ dedicates a container in order to facilitate the execution of such post-processing functions applying microservice principles, i.e., a certain function is executed when a particular request arrives. Developers can provide such functions during the deployment phase using two files. A codelet file provides a number of executable functions that use the outcome from one or more other functions and/or models. For each of these functions, a function file lists models and/or other functions, whose outputs are necessary for execution, as well as the type of its outcome. Using this file, we follow an event-triggered approach where execution of a function is prompted by the completion of these entities in the function file.

This architecture allows a multi-tenant model serving. The output from a single model execution can be used by multiple applications. For example, the output of an object detection model can be concurrently used by two applications that count the number of cars and identify the location of people, respectively. Critically, this output can be used to assist other tasks including collaborative sensing, without the need to instantiate a separate, dedicated object detection model. While SensiX++ makes it easier to deploy our collaborative sensing solution, it is not strongly decoupled, and can be served with other model serving mechanisms as well.

Our collaborative sensing system differs from SensiX [5] in that while all SensiX components are deployed in single edge devices, we move the post processing functions to a nearby, *edge cloud*. Moreover, we limit the scope of our work to vision models. The cameras are connected to devices powerful enough to run one or more models on the frames captured by the camera and pre-processing functions. On the other hand, post-processing functions can now collect model output from multiple observers to make an inference.

The architecture of the proposed collaborative system is described in Figure 1. In our case, Data Coordinator, Featurisation Coordinator, and Model Containers are located in the edge devices, whereas the query server and function coordinator are hosted in the Edge Cloud[1]. In addition, the Edge Cloud features other components that allow collaborative visual sensing including a ReID server to distinguish distinct objects in frames from different cameras and *View Mapping DB* that allows us to map object locations across various cameras.

In the rest of this section, we first explain how we achieve collaborative sensing through post-processing functions and describe these two components.

[1]Various other SensiX components are beyond the scope of this work, hence not explained in this paper.

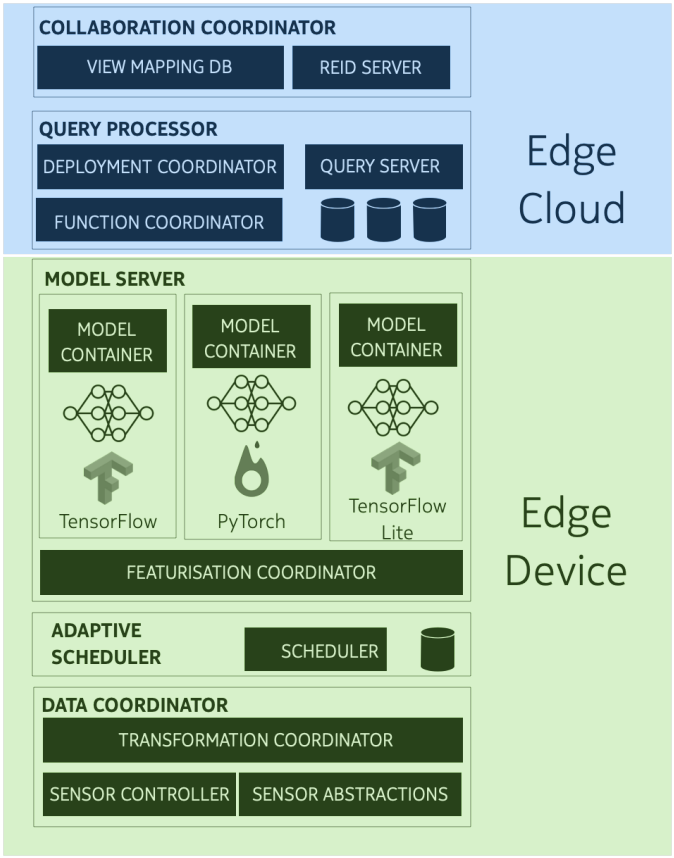


Fig. 1: Collaborative ML System Architecture

```
def count(**kwargs):
    detected_objects = kwargs.get('detected_objects')# Get detected objects
            from the model
    count = 0
    for obj in detected_objects:
        if obj.get('class_name') == kwargs.get('object_to_count'): # Check if
                the detected model is the desired object
            count += 1
    return count

def draw_boxes(**kwargs):
    image = kwargs.get('original_image') # Get the original image
    detected_objects = kwargs.get('detected_objects');# Get detected objects
    for obj in detected_objects:
        if obj.get('class_name') in kwargs.get('objects_to_draw'):
            bbox = obj.get('bounding_box') #Get the bounding box
            cv2.rectangle(image, bbox[0:1], bbox[2:3], 'red', 2) # Draw the
                    bounding box
            cv2.putText(image, obj.get('class_name'), (bbox[0], bbox[1] - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, 'red') # Write the label
    return image
```

Fig. 2: A code snippet for functions in the codelet

A. Function Execution

An important aspect of SensiX++ is that it allows developers to introduce custom functions that interpret the output from one or more models. These functions are introduced in *codelet* files during deployment. A codelet snippet is shown in Figure 2.

To serve query requests from users and applications, SensiX includes a microservice with a number of API endpoints. The APIs can be summarized as below:

`·/models`: Get the list of models

`·/functions`: Get the information about available post-processing functions

`·/inference/:function_id`: Get the outcome of a post-processing function

While functions can combine output from multiple models, all the models are hosted on the same device and the architecture assumes the device contains a single sensor for each sensing modality, i.e., a single camera, microphone, etc. On the other hand, we are interested in using the output of the same model running on different observers in addition to inferences using a single device. In order to facilitate this, the result of the `model` API provides the list of devices that each model runs. Similarly, the information returned by the `functions` API indicates if the function can only use the model output from a single observer or if it can combine results from multiple observers.

In addition, `inference` API includes an optional query parameter, `device_id`. If this parameter is not present, the function execution uses the data from all available observers. If there is a single `device_id` in the user request, the function execution is carried out exactly like that in SensiX. If the request includes multiple device ids, then the function execution combines the output only from those that are present in the query.

In order to facilitate function executions that use the model output from multiple observers, we introduce helper functions that the developers can use in their codelets that leverage a View Mapping DB. We now explain how this View Mapping DB is generated and queried. We will then describe how the View Mapping DB results can be used to answer queries that require collaboration across observers spatially and temporally.

B. View Mapping DB (Offline phase)

In many situations, multiple cameras are geo-distributed around a key area such as intersections of roads, malls, and airports to monitor the proximity from different view points. That is, there exist multiple observers viewing from different angles of a specific area. If video streams were to be analyzed individually without shared knowledge of the scene between the cameras to locate a suspicious target (e.g., the blue vehicle in Figure 3), video analytic services would have to run redundant object detection and re-identification on video frames where the target is already seen from other angles. To minimize redundant computation while providing a multi camera video analytic service, we create a spatial relationship between camera views and store this information in *View Mapping DB*.

View Mapping DB Population: A *View Mapping DB* matches a *specific area* of one camera's *field-of-view (FOV)* with other area on camera's FOV. To generate spatial relationship between the FOVs, it requires at least one unique object to be seen by one or more cameras. We leverage the ReID techniques to verify if there is a unique object seen at multiple camera FOV. This ReID service runs on a dedicated container in Edge Cloud and responds to ReID queries.

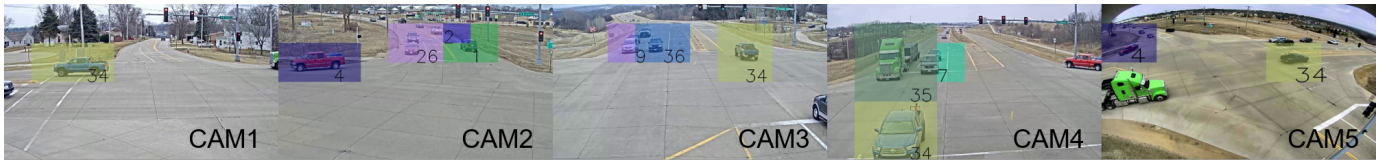


Fig. 3: Objects re-identified across multiple cameras are indexed with the same ID over other camera FOVs. For instance, blue pickup truck is labeled as ID 34 on 4 different camera FOVs.

During the offline phase, we collect video streams from multiple cameras and run an object detection model (e.g., Yolov3 [21]) on the video frame, which outputs the class, confidence score, and bounding box coordinate of each detected object seen. Each object detected on synchronized frames[2] are labeled with a *unique id*. For comparing the similarity of objects from multiple FOVs, features of detected objects are extracted using a feature extraction models (e.g., pixel color feature - RGB and HSI [22] or handcrafted feature - SIFT [23], SURF [24] or DNN features - OSNet [25]). These feature vectors from one camera are cross-compared with all object feature vectors in its neighboring cameras by using similarity metrics such as cosine similarity. If the similarity is over a threshold, it is s regarded as the same object across different camera FOV. Objects in the same camera cannot be regarded as the same object, since the same object cannot be present multiple times in the same FOV.

In case an object is verified as the same object, we can easily map a *specific area* of a camera with another. However, it would be impractical to match bounding box coordinates of both camera FOVs because the slightest difference in the coordinates would be recognized as different locations and, as a result, create an excessive number of entries which potentially is the same location in FOV. Therefore, we leverage the *tiling* technique which slices a camera's FOV into 10x8 tiles similarly done in [18]. The bounding box of the unique object is translated into minimum sets of tiles which cover the bounding box of the object. We leverage the AI City Challenge dataset [26] to showcase the View Map in Figure 3.

The View Mapping DB is a key-value data store. Each value stored in the data store refers to a collection of tiles across different cameras, as shown in Figure 4. In this example, if there is an object located on tiles 22, 23, 33, and 34 on camera 1, it is also located on tiles 26, 27, 37, and 38 on camera 5. The keys in the DB correspond to each of the entries in the listing, i.e., `1-[22, 23, 33, 34]`.

Once the spatial relationship is generated between cameras, it can be leveraged during runtime to search and match the region of interest on a camera to another. Similar to the ReID service, View Mapping DB also runs on a dedicated container and allows queries from other system components through API calls.

C. Identifying objects (online phase)

During the online phase, codelet functions introduced by the developers can utilize the View Mapping DB. A query to

[2]synchronized frames refer to video frames captured from multiple cameras at the same time

```
[
    {
        "camid": "1",
        "tiles": [22, 23, 33, 34]
    },
    {
        "camid": "5",
        "tiles": [26, 27, 37, 38]
    }
]
```

Fig. 4: An example value entry in View Mapping DB

the DB includes results from object/event detectors running on edge devices. Edge devices produce a list of objects, each of which consists of an object type and a bounding box. A query to the DB may include all or some of the objects reported by the edge devices, i.e., it is possible to filter objects using the object type. For each object, the database first extracts the tiles corresponding to the bounding box in a camera view and uses them to identify the common objects in different camera fields of view. For each object that the View Mapping DB returns, the information includes the bounding box in each camera. To give more freedom to the developers, the View Map service also provides an API that accepts a set of tiles and a camera id to return a list of tiles from the other cameras.

IV. Preliminary Results

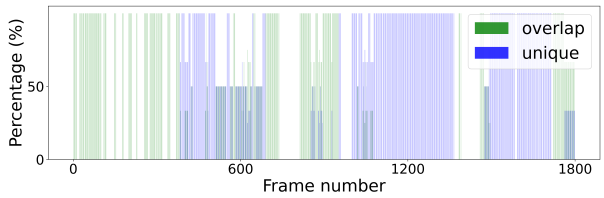
For our preliminary evaluation of the system, we evaluate the performance of the View mapping DB and discuss interesting findings. A summary of the preliminary results and observations is as follows:

- We observe that approximately over 50% of the objects detected in the dataset are seen from more than one camera FOV, which allows potential reduction of computation usage to detect vehicles on other camera FOV.
- View Mapping DB outputs with *only* spatial knowledge can be less accurate due to the different angle of the vehicles seen from one FOV to another.

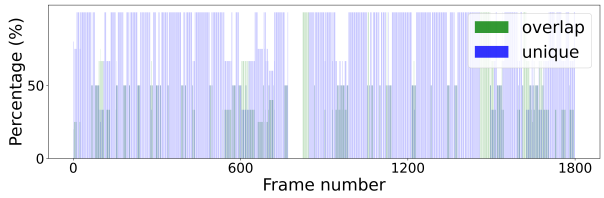
Note that a more comprehensive performance evaluation is left for future work.

A. Dataset

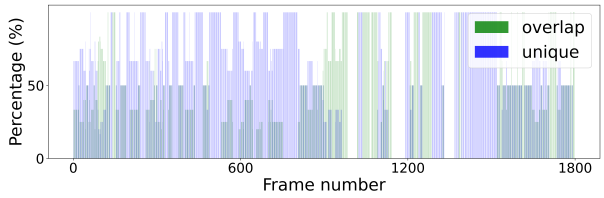
The AI City Challenge sector 1 dataset (AIC) [26] consists of 5 video streams, each recorded for 180 seconds (or 1800 frames) from different angles of an intersection. Each camera records at 1920×1080 (except camera 5, which records at 1280×960) and at 10 frames per second. Dataset provides



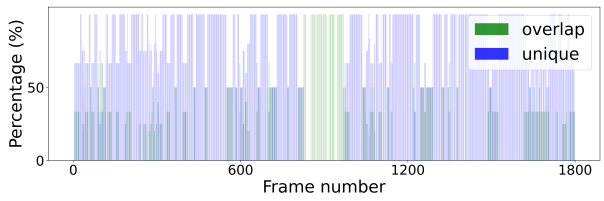
(a) Camera 1



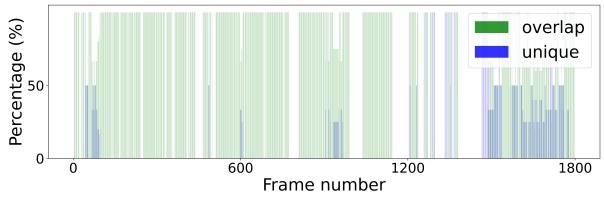
(b) Camera 2



(c) Camera 3



(d) Camera 4



(e) Camera 5

Fig. 5: Percentage of object's tiles seen from another camera's FOV over time. Green plots refer to object's tileset seen from at least two cameras, while blue plots refer to object's tileset uniquely seen from its own FOV.

information about only detected vehicles and contains approximately 100 unique vehicles which travel across at least two camera FOVs.

B. Evaluation1: View Mapping DB statistics

Using the AIC dataset, we first analyze the spatial relationship among cameras. Figure 5 shows the percentage of objects (tileset) in a camera, which overlaps with other camera views. The green bars indicate that some objects are seen from more than one camera FOV concurrently, while the blue bar indicates that objects are uniquely seen from its own camera. Regardless of the traffic patterns of the vehicles, cameras 1, 2, 3, and 4 have approximately 50% of the objects seen from multiple view points. However, in most cases, camera 5 tends to have an overlap between any other camera FOVs. Results demonstrate that there are regions for each camera's FOV (except 5) where objects are seen uniquely and thus View Mapping DB can provide insight in locating objects across camera FOVs.

C. Evaluation2: Queried result

For our second evaluation, we analyze the queried results of the View Mapping DB. The View Mapping DB contains approximately 7321 entries and each entry contains at least two or more distinct appearances across camera FOVs. We query the View Mapping DB with camera = 1, tile set = $[22, 23, 24, 32, 33, 34]$ (which references the green tiles on camera 1) to show the approximate location of vehicles across multiple camera FOVs. Figure 6a shows the result of the same query for a blue pickup truck and Figure 6b for a white minivan. While the result of the query shows that blue tiles capture both vehicles for camera 3 and 5, it is not accurate enough in camera 4. Notice that the size of the blue tiles in cameras 3, 4, and 5 are actually larger than the actual vehicle tiles in Figure 3 (with ID 34). This is because, even if both vehicles' query tile sets are equal, the posture of the vehicles and the distance from camera 1's FOV is slightly different.

While our current View Mapping DB is able to provide an approximate location of the vehicle across multiple camera views, only leveraging spatial knowledge can be less accurate. Therefore, we see some room for potential improvement by query optimization techniques to filter out tiles which cannot be candidate tiles. One approach is to slice the FOV of cameras to be more fine grain so that the query tiles do exhibit the blue pickup to be different from the white minivan. However, this approach can result in a smaller set of possible candidate tiles if the video feeds analyzed during the offline phase do not contain sufficient matches. Another addition to the View Mapping DB is applying temporal correlation between the cameras. That is, the View Mapping DB filters out less probable candidate tiles by view the tiles over a period of time. Naturally, vehicles cannot turn an angle from one direction to another between frames. Our future work involves both approaches to optimize View Mapping DB.

V. Discussion and Outlook

In this position paper, we present a work-in-progress collaborative sensing mechanism with multiple cameras at the edge. To achieve this, we deploy a View Map service that identifies common objects in different camera fields of view without extracting the object features at the run time. We provide an early evaluation of this service.

We are currently working on strategies to use View Mapping service output to facilitate temporal queries, e.g. object tracking across cameras in time. We plan to present a real life deployment of this system and its end-to-end evaluation in our future work.

(a) Queried location of blue pickup across multiple camera FOVs

(b) Queried location of white minivan across multiple camera FOVs

Fig. 6: This figure shows the result of a query to View Mapping DB. Green tiles on camera 1 refers to the queried tiles with tile set [22, 23, 24, 32, 33, 34]. Resulting tiles across cameras are shown in blue for camera 3,4,5.

REFERENCES

[1] 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.

[2] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, ser. IPSN '16. IEEE Press, 2016.

[3] F. Gustafsson, *Statistical sensor fusion*, 1st ed., 2010.

[4] V. Vielzeuf, A. Lechervy, S. Pateux, and F. Jurie, "Multi-level sensor fusion with deep learning," 2018.

[5] C. Min, A. Mathur, A. Montanari, U. G. Acer, and F. Kawsar, "Sensix: A platform for collaborative machine learning on the edge," *CoRR*, vol. abs/2012.06035, 2020. [Online]. Available: https://arxiv.org/abs/2012.06035

[6] R. Rajagopalan and P. K. Varshney, "Data-aggregation techniques in sensor networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 8, no. 4, pp. 48–63, 2006.

[7] 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.

[8] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *NSDI*, vol. 9, 2017, p. 1.

[9] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 389–400.

[10] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016, pp. 123–136.

[11] L. Zhou, H. Wen, R. Teodorescu, and D. H. Du, "Distributing deep neural networks with containerized partitions at the edge," in *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[12] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, 2020.

[13] 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.

[14] 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.

[15] 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.

[16] H. Qiu, X. Liu, S. Rallapalli, A. J. Bency, K. Chan, R. Urgaonkar, B. Manjunath, and R. Govindan, "Kestrel: Video analytics for augmented multi-camera vehicle tracking," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2018, pp. 48–59.

[17] H. B. Pasandi and T. Nadeem, "Convince: Collaborative cross-camera video analytics at the edge," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2020, pp. 1–5.

[18] 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," *arXiv preprint arXiv:2105.06524*, 2021.

[19] T. Zhang, A. Chowdhery, P. V. 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*. ACM, 2015, pp. 426–438.

[20] Y. Chang, A. Mathur, A. Isopoussu, J. Song, and F. Kawsar, "A systematic study of unsupervised domain adaptation for robust human-activity recognition," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 4, no. 1, Mar. 2020.

[21] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[22] K. R. Castleman, *Digital image processing*. CUMINCAD, 1993.

[23] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157.

[24] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[25] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-scale feature learning for person re-identification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3702–3712.

[26] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, L. Zheng, A. Sharma, R. Chellappa, and P. Chakraborty, "The 4th ai city challenge," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020, p. 2665–2674.