Multi-View Scheduling of Onboard Live Video Analytics to Minimize Frame Processing Latency

Shengzhong Liu[†], Tianshi Wang[†], Hongpeng Guo[†], Xinzhe Fu[‡],
Philip David[§], Maggie Wigness[§], Archan Misra[*], Tarek Abdelzaher[†]
[†]University of Illinois at Urbana-Champaign, [‡]Massachusetts Institute of Technology,
[§]US Army Research Labs, [*]Singapore Management University
Email: {sl29, tianshi3, hg5}@illinois.edu, xinzhe@mit.edu,
{philip.j.david4.civ, maggie.b.wigness.civ}@mail.mil, archanm@smu.edu.sg, zaher@illinois.edu

***Abstract*—This paper presents a real-time multi-view scheduling framework for DNN-based live video analytics at the edge to minimize frame processing latency. The work is motivated by applications where a higher frame rate is important, not to miss actions of interest. Examples include defense, border security, and intruder detection applications where sensors (in this paper, cameras) are deployed to monitor key roads, chokepoints, or passageways to identify events of interest (and intervene in real-time). Supporting a higher frame rate entails lowering frame processing latency. We assume that multiple cameras are deployed with partially overlapping views. Each camera has access to limited onboard computing capacity. Many targets cross the field of view of these cameras (but the great majority do not require action). We take advantage of the spatial-temporal correlations among multi-camera video streams to perform target-to-camera assignment such that the maximum frame processing time across cameras is minimized. Specifically, we use a data-driven approach to identify objects seen by multiple cameras, and propose a batch-aware latency-balanced (BALB) scheduling algorithm to drive the object-to-camera assignment. We empirically evaluate the proposed system with a real-world surveillance dataset on a testbed consisting of multiple NVIDIA Jetson boards. The results show that our system substantially improves the video processing speed, attaining multiplicative speedups of $2.45\times$ to $6.85\times$, and consistently outperforms the competitive static region partitioning strategy.**

***Index Terms*—Edge Computing, Live Video Analytics, Collaborative Sensing**

I. Introduction

This paper introduces a scheduling framework that optimizes the frame processing latency for DNN-based analytics of live video streams associated with a multi-camera infrastructure deployment, where an overall region is collectively monitored by a set of cameras with partial spatial field-of-view (FoV) overlaps (*e.g.*, CityFlow [1]). One such example is shown in Figure 1. The work is motivated by defense and security applications that push an increasing amount of autonomy to the edge. For example, in a future conflict, cameras might be deployed around key roads and chokepoints to identify unauthorized or trespassing targets. The areas surveyed by these cameras might see a lot of traffic. Only targets of interest require immediate responding actions. The decision of whether a target is "of interest" requires processing the video by a DNN (such as YOLO) to identify target type. Faster frame processing speed not only improves the object recognition and tracking fidelity, but also helps reduce the end-to-end system response delay to physical events.

Centralized video processing approaches [2]–[5], where camera video streams are transferred for DNN execution to a powerful cloud server, impose significant bandwidth and energy overheads and also suffer from higher communication latency and privacy concerns. Accordingly, we consider a decentralized computing paradigm, where the captured high-quality video frames are processed by each individual camera node, using limited onboard computing capacity, likely including a lower-end GPU. Our general idea is to reduce the DNN execution latency by eliminating the *redundant* DNN pipeline inspections, on multiple cameras, on such overlapping regions. Recent approaches [6], [7] for eliminating or minimizing such redundant DNN inspections have utilized a static, spatial partitioning approach to turn off some of the cameras. In contrast, we propose a finer-grained *object-level, workload-aware, latency-balancing* approach, where the responsibility for tracking distinct objects in the overall shared region, is dynamically and periodically redistributed among the set of collaborating cameras. It is motivated by two key limitations observed with the static approaches:

- The overall frame processing latency of an individual camera, proportional to the number of objects it currently tracks, shows significant variability across time. As an illustration, Figure 2 plots the number of objects/frame (sampled once every 2 secs) across the FoV of 5 distinct cameras in Figure 1: both the absolute workload of individual cameras, and the relative workload between camera pairs show significant temporal variation.
- Due to the opportunistic, progressive nature of such in-the-wild infrastructure deployments, there is often significant heterogeneity in the processing capability across cameras– *e.g.*, the memory capacity and GPU cores available. As a result, an identical object-level workload may generate very different processing latency on distinct devices.

Our proposed approach for low-latency, distributed execution is built on two inter-related concepts.

Low-latency DNN execution. First, we reduce the average

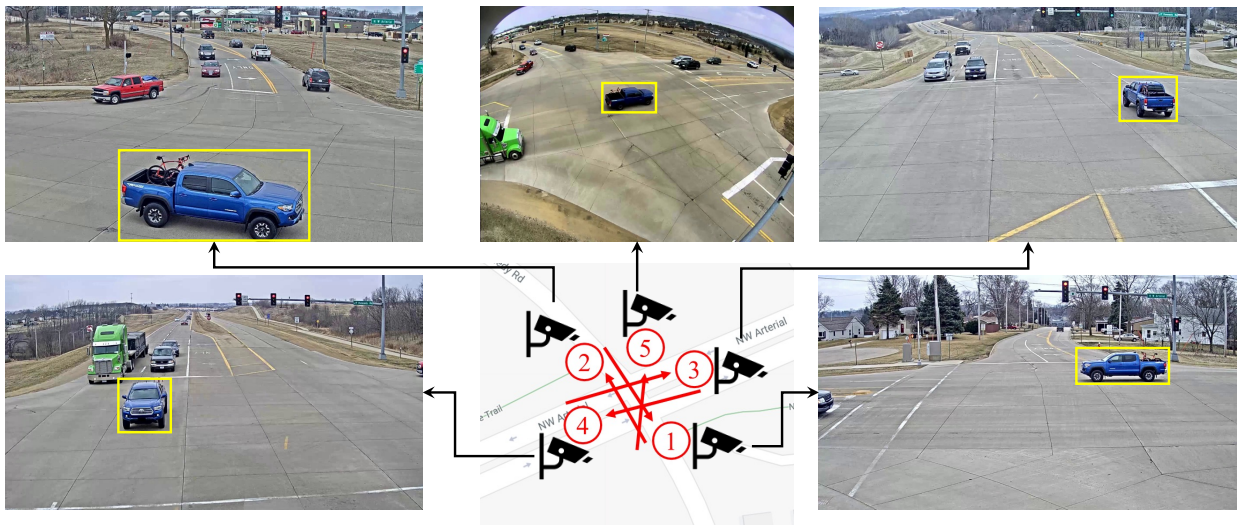


Fig. 1: An example of camera view overlaps. We use yellow boxes to highlight an object observable to every camera.

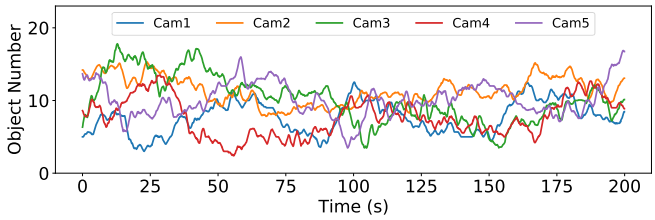


Fig. 2: Temporal variation of object workload across cameras.

computation load by running full-frame DNN inspection (a comparatively high latency task) only sporadically, *e.g.*, once per second, and use motion flow-based estimation techniques to confine the DNN inspection on remaining frames to only extracted salient regions (an approximate area around each detected object that the camera is responsible for). Figure 3 illustrates this mechanism of full-vs.-partial frame inspection. It eliminates the needs for repetitive DNN scanning on background regions and reduces the frame processing latency to a degree that is proportional to the number of tracked objects. Second, we accelerate the DNN inference on GPUs by leveraging the task batching mechanism [8], [9] (see Figure 4) to process multiple confined partial frames with the same size in parallel on the GPU, thereby achieving notably lower latency than the traditional serialized processing approach.

Dynamic latency-balancing scheduling. We dynamically partition the tracking workloads of objects in the overlapped regions among cameras to balance their frame processing latency under batched execution. Different from the static partitioning strategy, our approach performs a *load-and-resource aware* assignment of object tracking responsibilities across the camera nodes, taking both the hardware processing capacity and relative object tracking workload, into the consideration. It minimizes the processing latency on heavy-loaded cameras, by skipping tracking objects in the overlapping regions, which are instead tracked by light-loaded cameras.

Specifically, we formulate the scheduling problem that decides the object-to-camera assignment as a multi-view scheduling (MVS) problem with task batching. It seeks to minimize the maximum processing latency - the time it takes to process a frame - across all cameras. We prove the MVS problem to be strongly NP-hard, and correspondingly propose a batch-aware latency-balanced (BALB) scheduling algorithm to approximately solve it.

The BALB algorithm employs a hybrid centralized-cum-distributed scheduling mechanism to intelligently utilize the observed spatial redundancy to reduce and distribute the object tracking workload (latency) across the collective set of camera nodes. First, all cameras communicate with a *central scheduler*, which can be an edge node. After a full frame inspection, each camera uploads its list of detected objects to the central scheduler. The central scheduler first associates the detected bounding boxes and identifies the common objects across cameras, and then runs a *central stage* of the BALB algorithm to derive an initial object-to-camera assignment. The individual computing capacities and inspection workload of cameras, as well as the task batching opportunities are all considered in the optimization.

To additionally tackle the unforeseen object dynamics between such periodic assignments (such as a new object entering the monitored region), in between full-frame inspections, each camera independently runs a *distributed stage* of the BALB algorithm to decide whether to track a new object or not. To minimize the need for frequent, per-frame communication, this distributed scheduler employs spatial partitioning to update the assignment for new objects and apportions the responsibility for tracking such object dynamics.

We implement the proposed scheduling framework on a testbed consisting of multiple heterogeneous NVIDIA Jetson (Nano, TX2, Xavier) boards. We comprehensively evaluate the system performance using the AI City Challenge 2021 (AIC21) multi-camera video surveillance dataset [1], [10]. The results show that our system consistently improves the frame processing speed by by $2.45\times$ to $6.85\times$, at the cost

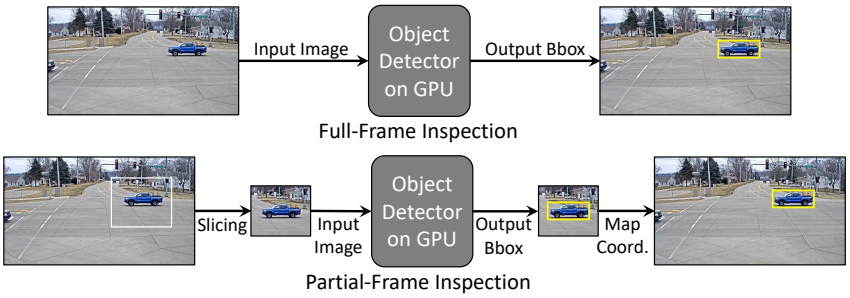


Fig. 3: Full-frame inspection vs. partial frame inspection.

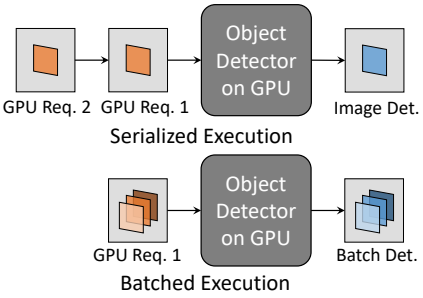


Fig. 4: Serialized vs. batched exec.

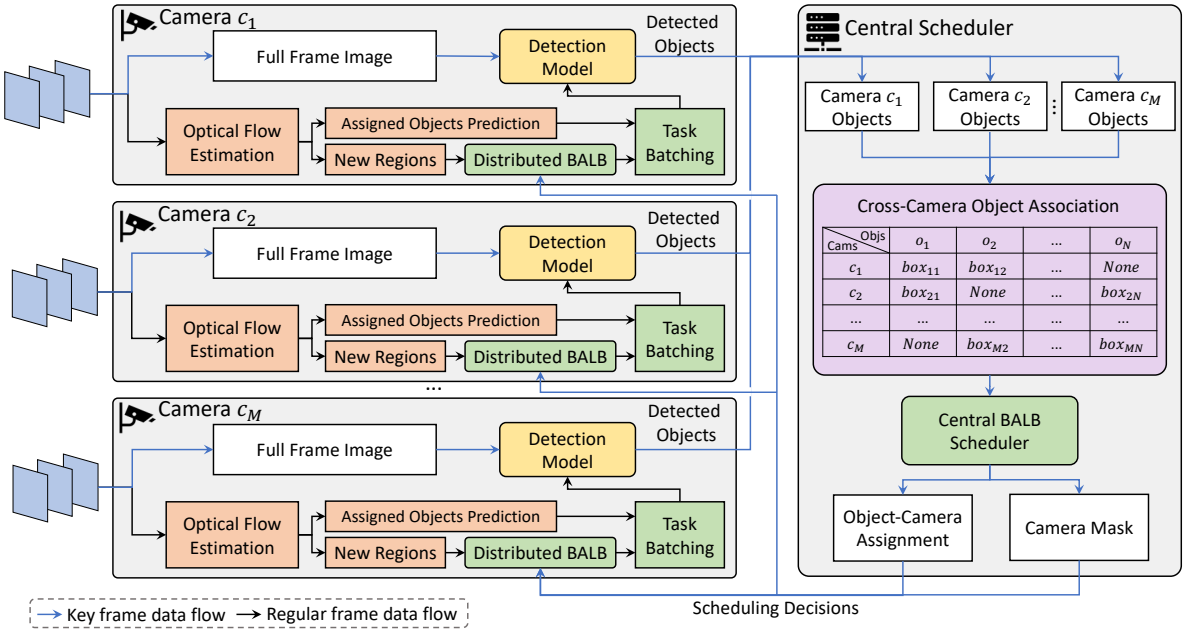


Fig. 5: Framework overview. For the block colors: The tracking-based slicing module is highlighted in orange, the cross-camera association module is shown in purple, and the BALB scheduler related operations are highlighted in green.

of minor degradation on detection quality. We also show that this dynamic, object-level scheduling approach is superior to a competitive static spatial partitioning baseline, providing an average of $1.88\times$ reduction in processing latency.

The rest of this paper is organized as follows: We first give an overview of the proposed framework in Section II. In Section III, we formulate the scheduling problem and introduce the proposed BALB algorithm. Evaluations results are presented in Section IV. After discussing the limitations in Section V and reviewing the related literature in Section VI, we conclude the whole paper in Section VII.

II. System Architecture

In this section, we first give a brief overview of the proposed architecture, then introduce the individual components that constitute our system.

A. System Overview

Assume we have a set of static cameras deployed around a local area, such as a shopping mall or a traffic intersection. The cameras share complementary views to obtain comprehensive perception of the target area. They capture image frames regularly at the same frequency, *e.g.*, 10FPS. Further, their views are partially overlapped so one object may be simultaneously observable from multiple cameras. One such example is given in Figure 1. Each camera is equipped with limited computing capacity, *e.g.*, low-end GPUs, so they can run object detection and tracking locally. Each object needs to be localized, classified, and tracked at every frame. Since deep neural detection models, like YOLO [11], are resource consuming, we regard them as the main workload to optimize. The heterogeneous computing resources among cameras might lead to different DNN processing speed. The detection model accepts input images with various spatial sizes and batch sizes (*i.e.*, number of images in a batch), where smaller images achieve shorter execution latencies. Only the images with the same spatial sizes can be put into a single batch.

The objective of the scheduling is to achieve real-time live video analytics on cameras under limited *computing capacity* and *network bandwidth*. We assume that, on one hand, the onboard GPUs are not powerful enough to process every full camera frame in real-time, because running the detection model on the full frames takes longer than the camera sam-

pling interval. On the other hand, the cameras do not have enough network bandwidth to stream high-resolution videos to the cloud in real-time for centralized processing.

We optimize the analytics efficiency by taking advantage of the spatial-temporal correlations in multi-view video streams to reduce the workload, and by applying effective task batching on GPUs to accelerate the DNN inference. An overview of the proposed system is given in Figure 5. Instead of running full-frame inspection on every sampled image, we only run it on some frames (which we call the *key frames*, blue arrows in Figure 5) at a fixed low frequency (*e.g.*, once per second). At the remaining frames (which we call the *regular frames*, black arrows in Figure 5), we run a set of partial-frame inspections to search the exact object locations around their approximately predicted locations. The period of frames between two key frames is called a *scheduling horizon*. When an object appears in multiple cameras, we only schedule one camera to track it. We refer to the scheduling decision about the subsets of objects each camera track as the *object-camera assignment*, which should consider both the latency balancing among cameras and GPU task batching opportunities at each camera. It is comprised of three main modules:

- **Tracking-based image slicing:** At regular frames, we slice frames into partial regions around predicted object locations, according to the projection from a tracking algorithm, and the regions outside the regions of interest are not inspected by the DNN.
- **Cross-camera object association:** We associate individually detected object bounding boxes across cameras to identify the common objects, and further reduce redundant tracking of the same object.
- **Multi-view scheduling:** We decide the subset of objects to track by each camera and their corresponding batching decisions, to minimize the maximum camera latency.

B. Optical Flow-based Tracking and Image Slicing

We apply the optical flow-based tracking algorithm in [12]. It follows a general tracking-by-detection paradigm [13], where object tracking is achieved by first detecting all objects in the new frame and then associating them with previously tracked object trajectories based on the location overlaps [14], where applicable. The approach uses an optical flow model [15] to predict the current locations for previously detected objects. Optical flow estimates the pixel motions between two input images. Optical flow-based tracking effectively combines the object location information in the previous frame and the pixel motion information extracted from the new frame. Since the input images in a batch should have the same spatial size, the predicted object locations are expanded to the nearest size in a quantized set to increase the batching opportunities. The quantized size is fixed for each object within a scheduling horizon, and downsizing is performed if the object size grows beyond it.

Besides, the estimated pixel motions can be used to roughly identify newly appeared objects at regular frames. Since we assume the cameras are statically mounted, the pixel motions in optical flow are purely caused by the object movements. In this paper, we define the clusters of moving pixels that do not belong to any predicted bounding box for existing objects as a *new region*, where a new object may appear. We also feed these regions into the object detection model to detect newly appeared objects. By doing so, we are able to detect new objects at their first appearance, instead of waiting until the next full-frame inspection.

C. Cross-Camera Object Association

We next explain how we perform cross-camera object associations. The objective of this module is to identify common objects that appear in the view of multiple cameras. Specifically, the algorithm associates a detected object by one camera to the detected objects by the remaining cameras. We only consider pixel location-based association approaches, because semantic feature-based association approaches are typically complicated which can not run in real-time on resource constrained cameras. As the angles of camera views might present huge difference (as shown in Figure 1, the differences of angles among camera 1, 2, 3, 4 are 90° or 180°), it is hard to use conventional vision techniques such as homography transformation to directly map the pixel locations of an object from one camera to another camera. However, since the cameras are statically mounted, the mapping relations of objects among cameras are fixed, except that the size (width, length, and height) and facing direction of objects can change[1]. Motivated by [16], we design a data-driven approach to fit the location mapping relations between the cameras.

As the detected object location is represented by a pixel-level rectangular bounding box, we use P_{ij} to denote the bounding box of the j-th object on camera c_i. To calculate the corresponding bounding box of P_{ij} on camera i', our method works in three steps. First, we run a classification model to determine whether P_{ij} appears in the view of camera $c_{i'}$. If the prediction is negative, we terminate this process. Otherwise, we move to the second step. Second, we use a regression model to predict the mapped location of P_{ij} on camera i', denoted as $\tilde{P}_{ij}^{i'}$. For both classification and regression models, we use a non-parametric K-Nearest Neighbors (KNN) model. It works as a special lookup table which uses the nearest case(s) in the memory to generate the prediction. Third, having $\tilde{P}_{ij}^{i'}$ as the predicted location of P_{ij} on camera $c_{i'}$, we then calculate the proximity between $\tilde{P}_{ij}^{i'}$ and all the detected objects on camera $c_{i'}$ based on their area overlaps (*i.e.*, intersection over union). We run a Hungarian algorithm to find the most proximate detected object on camera $c_{i'}$ to $\tilde{P}_{ij}^{i'}$. The bounding box $P_{i'j'}$ with the maximum proximity and within a preset threshold will be taken as the matching of P_{ij}. At this point, the detected object $P_{i'j'}$ at camera $c_{i'}$ and P_{ij} at camera c_i are taken as the same object. The association procedure is illustrated in Figure 6.

Assume we have a list of M cameras in total, we iterate over the list, and run the cross-camera object association for camera

[1]These 3D factors can lead the mapping of 2D bounding boxes between cameras to be non-linear, which makes homography fail.

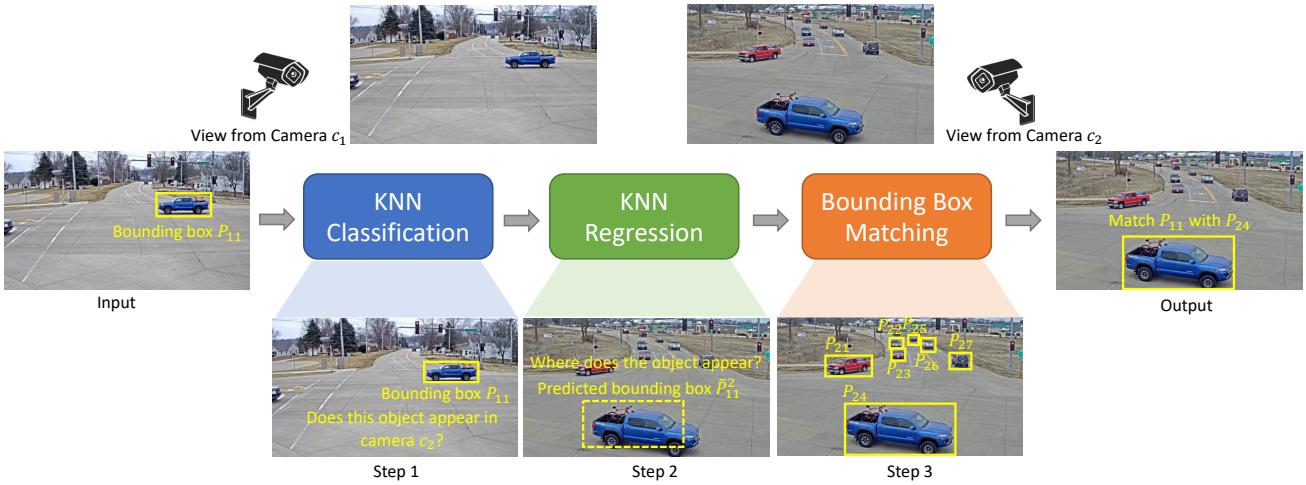


Fig. 6: Cross-camera object association. To match P_{11} from camera c_1 to camera c_2, the algorithm runs three steps: (1) We use a k-nearest neighbors classification (KNN-classification) model to determine whether P_{11} appears on camera c_2. (2) We use another KNN-regression model to predict the estimated pixel locations of the object on camera c_2, which is $\tilde{P}_{11}^2$. (3) We calculate the proximity of all the detected bounding boxes on camera c_2 to $\tilde{P}_{ij}^k$. Finally, P_{11} is matched with P_{24}.

c_i with every camera $c_{i'}$ behind it in the list, *i.e.*, $i' > i$. A round of cross-camera object association terminates when all the detected objects by each camera have been matched with the detected objects on all remaining cameras. The associated object list will be passed to the multi-view scheduling module.

However, to train the aforementioned classification and regression models in offline, we need to first collect a supervised training dataset with human labels associating same objects across cameras for each deployment scenario, such that the knowledge can be passed to the machine learning models to recognize whether the object in one camera also appears in another camera, and where it appears. The estimated camera poses and their spatial correlations are automatically encoded in the two models. It only works with static camera deployment, and will require the model retraining when the camera pose changes. We acknowledge the human effort involved in the labeling process, and list it as one of our future works to achieve the automatic classification/regression model training. For example, during the training stage, we can utilize the prediction from a reliable semantic-based ReID model [17] as the groundtruth labels to train the lightweight location-based models for real-time deployment.

D. Multi-View Scheduling

After we obtain the associated object list, the remaining problem becomes how to assign objects to cameras for tracking, such that the maximum execution time among cameras is minimized. We formulate it as a multi-view scheduling (MVS) problem, and correspondingly propose a two-stage batch-aware latency-balanced (BALB) scheduling algorithm to solve it. It considers both the latency balancing among cameras and the task batching opportunities on GPU. The cameras track and batch the assigned objects according to the scheduling decisions. The details will be explained next.

III. Multi-View Scheduling

In this section, we first describe the task execution model, and then formulate the *multi-view scheduling (MVS) problem*. Finally, we introduce the *batch-aware latency-balanced (BALB) algorithm*, to solve the formulated MVS problem.

A. Object Detection Task Model

Assume we have a camera set $\mathcal{C} = \{c_1, c_2, \ldots, c_i, \ldots c_M\}$ consisting of M cameras equipped with heterogeneous computing capacities. They monitor a local area with overlapped views, and run object detection models (*e.g.*, YOLO [11]) (along with a lightweight tracking algorithm) to track the appeared objects. A fixed number of T frames are captured within a scheduling horizon, including a key frame and $T-1$ regular frames. At start of the scheduling horizon, a set of N physical objects $\mathcal{O} = \{o_1, o_2, \ldots, o_j, \ldots, o_N\}$ are detected from the latest full-frame inspection, and their locations on each camera are identified (if appear) after running the cross-camera object association on a central scheduler. However, the object set $\mathcal{O}$ may evolve during a scheduling horizon since new objects may arrive and existing objects may leave. We want to track the locations of all appeared objects, through scheduling partial-frame inspection tasks on the cameras. We define the *coverage set* $\mathcal{C}_j \subseteq \mathcal{C}$ of an object o_j as the subset of cameras that can see it. The object can be tracked from any camera $c \in \mathcal{C}_j$.

Each object o_j is associated with a target size s_{ij} at each camera $c_i \in \mathcal{C}_j$, which defines the size of the partial regions where we will search the object. The target size for the same object can be different among cameras, but it is fixed in a scheduling horizon (with possible downsizing) on the same camera. Since we can only batch input images with the same size on GPUs, we quantize the target sizes (by expanding the regions) to a limited set $\mathcal{S} = \{s_1, ..., s_K\}$ to increase batching

opportunities. Given a target size s, at most B_i^s partial regions can be batched and processed in parallel on camera c_i, which is called the *batch limit* of target size s on c_i, and the incurred latency is t_i^s, correspondingly[2]. The batch limit for the same target size can be different across cameras.

B. Scheduling Problem Formulation

We assume partial regions corresponding to different frames are processed sequentially without cross-frame batching. Here we focus on formulating the scheduling problem for a single frame. Before that, we first define the camera latency below.

Definition 1 (Camera Latency)**.** *Given a camera c_i, its latency L_i is defined as the sum of execution latencies of all its batches belonging to one frame. No preemption is allowed during batch executions.*

We further define the *system latency* L as the maximum latency among all cameras in set $\mathcal{C}$, *i.e.*, $L = \max_i L_i$. The scheduling problem we study is to derive a *feasible assignment* $\mathbf{X}$ between cameras and objects such that the system latency is minimized. The feasible assignment is formally defined below.

Definition 2 (Feasible Assignment)**.** *An assignment between a set of cameras $\mathcal{C}$ and a set of objects $\mathcal{O}$ is a matrix $X \in M \times N$, where $x_{ij} \in \{0, 1\}$ indicates whether camera c_i tracks object o_j. An assignment is feasible if it satisfies two conditions: (1) Each object is tracked by at least one camera that can see it, i.e., $\sum_{c_i \in \mathcal{C}_j} x_{ij} \geq 1, \ \forall j$. (2) No object can be tracked by a camera that can not see it, i.e., $\sum_{c_i \in \mathcal{C} \setminus \mathcal{C}_j} x_{ij} = 0, \forall j$.*

Given a feasible assignment, it would be trivial to convert it to the optimal batch sequences at each camera to achieve the minimum latency. The camera latency only depends on the number of batches for each target size. We derive the optimal batch sequence on each camera by batching objects with the same target size in a greedy manner, which apparently minimizes the number of used batches. Each target size is independently batched. Thus, a feasible assignment uniquely decides the corresponding optimal system latency. We then formally define the multi-view scheduling (MVS) problem below.

Definition 3 (Multi-View Scheduling Problem)**.** *The Multi-View Scheduling (MVS) problem asks for a feasible assignment between camera set $\mathcal{C}$ and object set $\mathcal{O}$ such that the system latency L is minimized.*

We establish the computational complexity of the MVS problem in Claim 1.

Claim 1. *The MVS problem is strongly NP-hard.*

Proof. We prove by reducing the bin packing problem to the MVS problem. We first constraint the MVS problem to an identical machine scheduling (IMS) problem by adding the following constraints: 1) The batch limit is always one (*i.e.*, no batching is allowed); 2) Every object can be seen from all cameras, thus can be assigned to any camera; 3) All cameras have the identical processing speed; 4) Each object has the same target size across all cameras, so its execution latency is the same on different cameras. Minimizing the system latency in the IMS problem can be converted to an equivalent decision problem: Given a time budget T, can we finish processing assigned objects on all M cameras? If we consider M identical cameras as M bins with capacity T and regard the N objects as N items with their execution latency defined as the item size, then the decision version of assigning objects to cameras becomes a standard bin packing problem, which has been proved to be strongly NP-hard [18]. The claim follows. □

We next propose an efficient algorithm that approximately solves the multi-view scheduling problem.

C. Batch-Aware Latency-Balanced Scheduling Algorithm

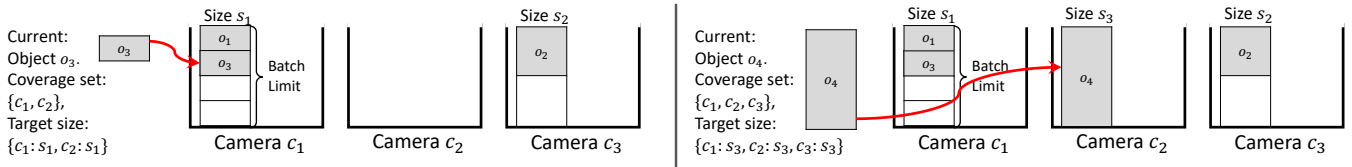
The scheduling decisions can not be made statically at offline, because the number of appeared objects dynamically change the latency of cameras at runtime, which further affects the optimal object-camera assignment. In addition, the object set $\mathcal{O}$ may evolve within a scheduling horizon, making the scheduling more challenging. A camera is unaware of new object arrivals at other cameras unless the new object appears in a region that both cameras can see it simultaneously. From the optimization perspective, the updated object list at cameras should be uploaded to the central scheduler at every frame, to produce the updated assignment. However, too frequent camera-scheduler communication may slow down the processing significantly. From the scheduling efficiency perspective, we should design a fully distributed mechanism which runs independently at each camera, so that no waiting happens at cameras. However, it ignores the unbalanced latency among cameras, and will produce inferior assignment.

We observe that objects appear and disappear at a low frequency compared to the camera sampling frequency [12]. We thus propose the **Batch-Aware Latency-Balanced (BALB)** scheduling algorithm that works in two stages, as illustrated in Figure 5. We run a *central stage* on the central scheduler at each key frame to produce an initial assignment based on the associated object list, and then run a *distributed stage* independently on each camera at each regular frame to update the assignment based on the change of appeared object set. The potential imbalance produced by the distributed stage will be corrected by next central stage in a few frames.

1) Central Stage: After a full-frame inspection, the cameras upload the list of their detected objects to the central scheduler. The central stage algorithm takes the associated object list as input to produce the initial assignment. It works in a batch-aware latency-balanced manner to fully exploit the camera view overlaps and task batching mechanisms on GPUs.

We initialize the camera latency as their corresponding full-frame inspection time t_i^{full}, and then minimize the maximum

[2]Although batching too many images would lead to a non-ignorable increase in execution latency, we operate in a region where the execution time changes only slightly with batching (before an inflection point is reached where the slope increases). We correspondingly set the execution time at the batch limit as the batch execution latency t_i^s.



Step 3: We assign o_3 to c_1, because there is an incomplete batch of size s_1.

Step 4: We assign o_4 to c_2 , because it has the min load after including o_4 .

Fig. 7: In BALB central stage, we use examples to explain when we assign an object to an incomplete batch and when we need to start a new batch. For simplicity, we assume each object has the same target size at each camera.

Algorithm 1: Central Stage BALB Algorithm

Input: Object coverage sets $\mathcal{C}_1, \ldots, \mathcal{C}_N$, camera execution latencies t_i^s, and batch limits B_i^s, $\forall s \in \mathcal{S}, \forall c_i \in \mathcal{C}$.

Output: Feasible assignment $\mathbf{X}$, camera latency L

Initialize: $x_{ij} := 0,\ \forall i, j,\ L_i := t_i^{full},\ \forall i$;

Reindex the objects $o_j \in \mathcal{O}$ by non-decreasing order of $|\mathcal{C}_j|$ (ties broken in favor of larger target size);

for $o_j \in \mathcal{O}$ *(after object reindexing)* **do**

  $\mathcal{C}'_j := \{c_i | c_i \in \mathcal{C}_j$ **and** $\exists$ incomplete s_{ij} batch$\}$;

  if $|\mathcal{C}'_j| > 0$ **then**

    $c_{i^*} :=$ The camera in $\mathcal{C}'_j$ with the largest relative capacity in the incomplete s_{i^*j} batch;

    $x_{i^*j} := 1$;

  end

  else

    $c_{i^*} := \arg\min_{c_i \in \mathcal{C}_j} L_i + t_i^{s_{ij}}$;

    $x_{i^*j} := 1,\ L_{i^*} := L_{i^*} + t_{i^*}^{s_{i^*j}}$;

  end

end

Return the assignment $\mathbf{X}$ and the camera latency L;

camera latency through maintaining a good latency balancing property among cameras during the process. We try to assign each object to a camera in a single pass. The idea is motivated by the following observation: For an object o_j, the more cameras can see it, the more flexibility we have in scheduling its tracking. Therefore, we start with assigning objects that are observable from only one camera, which have a deterministic assignment. After that, we gradually assign objects with more flexibility (*i.e.*, larger coverage sets). When assigning an object o_j, we try to maximally batch tasks. We will not start a new batch for o_j as long as there exists a camera $c_i \in \mathcal{C}_j$ that has an incomplete batch (*i.e.*, the batched image count is below the batch limit) for target size s_{ij}. If there are multiple cameras with incomplete batches, we choose the one with the maximum *batch capacity*, which is defined below.

Definition 4 (Batch Capacity)**.** *Given an incomplete batch with b batched images, the batch capacity is defined as $BC = B - b > 0$, where B is the corresponding batch limit.*

Otherwise, we have to start a new batch for o_j. In this case, we select the camera that has the minimum updated latency after including the new batch. It is different from assigning

Fig. 8: The camera masks. We assume the (increasing) latency-based camera order is: $c_3 > c_1 > c_2$. Each camera only tracks new objects at cells that are unobservable from higher priority cameras. If we regard the blue vehicle as a new object, camera c_1 will track it.

o_j to the camera with the minimum current latency because the inspection latency for the same object may be different among cameras due to different target sizes and heterogeneous GPU speed at cameras. The details of the proposed central stage BALB algorithm are summarized in **Algorithm 1**, and an illustrative example is given in Figure 7. It has a low computation complexity as $\max(O(N \log N), O(MN))$. One limitation in the current BALB algorithm is that we do not consider whether one camera may be able to track the same object for a longer period that other cameras, according to the object moving patterns.

2) Distributed Stage: The distributed stage independently runs at each camera to update the assignment for newly appeared objects and objects that disappear on their originally assigned cameras. The assignment of the remaining objects are unchanged. The objective of this stage is to first guarantee that each appeared object is tracked by at least one camera, and then optimize the efficiency as much as we can.

Algorithm 1 can not be directly used because the cameras do not know the exact latency and task batching conditions on other cameras. Although the batch limit on each camera is known in advance, the object arrival and leaving information is not shared among cameras, so we can not consider batching in the distributed stage scheduling. To make sure the cameras make consistent decisions under no communications, we have to rely on fixed policies that work in a self-organized way. Specifically, we sort the cameras in increasing order of their assigned latency by Algorithm 1, and use that order as the fixed camera priority to assign objects. This order is determined at the central stage and will be fixed during the scheduling horizon. Each camera only tracks new objects at regions that are unobservable from all higher priority cameras.

We apply the policy to two situations. First, we define new

objects as objects that arrived after the last key frame. New objects enter the view of different cameras asynchronously. Therefore, we dynamically update their assignment at each frame. Specifically, we choose the camera from its coverage set that has the highest priority to track it at each frame. Second, existing objects exit the view of different cameras asynchronously. When an object leaves its assigned camera, other cameras that can see it should take over its tracking. Specifically, at each camera, for each existing object that was not assigned to it, we test whether the object has disappeared on its assigned camera but is still observable from this camera. If yes, we select the camera with the highest priority from its new coverage set to take over its tracking. This decision is made automatically at each camera without cross-camera communication, because they base on the synchronized information (*i.e.*, cell masks) to make the decision. The computation complexity of the distributed-stage scheduling algorithm is only $O(N)$.

In both cases, the camera will automatically start tracking the object if itself is selected, without communication cost. In the implementation, we compute a mask for each camera after the central stage, as shown in Figure 8, which indicates regions where the camera should track the new objects. We first divide the camera frame into a grid of pixel-level cells, and compute the coverage set for each cell. The computation of the coverage set for each cell relies on the cross-camera classification and regression models, thus only works with static camera poses. The approach combines the information of camera coverage overlaps with the current estimation of camera load. We then choose the camera with the highest priority to cover the cell. In the distributed stage, the new objects, or objects that exit their assigned cameras, are automatically assigned according to the camera masks.

IV. Experiment

In this section, we evaluate the proposed scheduling framework on a testbed consisting of various Jetson models with the AI City Challenge 2021 (AIC21) dataset.

A. Implementation and Experimental Setup

1) Testbed: We implement the system on a heterogeneous edge testbed consisted of 5 NVIDIA Jetson devices: 2× Jetson Xavier, 2× Jetson TX2, and 1× Jetson Nano. Figure 9 shows the hardware platform we used. Each device corresponds to a smart camera. They are deployed in an off-campus building, and connected to the central scheduler through wired network (100Mbps Downlink, 20Mbps Uplink). We deploy the central scheduler to a desktop with Intel i9960x CPU located in a campus building. We did not explicitly experiment with wireless network in this paper and leave it as a future investigation. We utilize TCP socket programming for reliable data communication between the edge devices and the central scheduler.

2) Dataset: We evaluate the system with AI City Challenge 2021 dataset published by NVIDIA [1], [10]. It consists of traffic camera data collected around traffic intersections and streets. We choose three deployment scenarios to run the experiment. Among the three scenarios, S1 has 5 cameras mounted around a traffic intersection facing different directions of traffic where regular traffic patterns are observed caused by the traffic lights; S2 has 2 cameras mounted at a roadside of residential area, where the vehicles only distribute sparsely; S3 has 3 cameras in total where 2 cameras monitor a fork road and 1 camera faces the roadside, and busy traffic is observed. The dataset provides synchronized videos from each camera with a frame rate of 10 fps. The full image resolution we use is 1280×704 for regular cameras, and 1280×960 for fisheye cameras. For each camera, we use half length of the video to train the cross-camera object association model with the provided bounding box labels, and use the remaining half for testing. The specific edge device configuration for each evaluation scenario is listed in Table I.

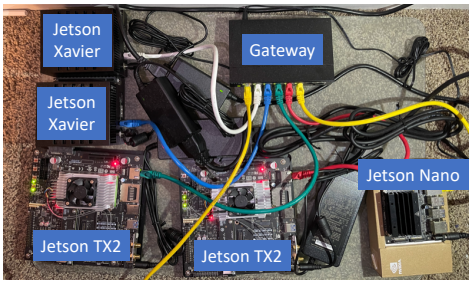


Fig. 9: Heterogeneous edge testbed.

TABLE I: Hardware Configuration for Each Scenario

Scenario	Edge Device Configuration
S1	Jetson Xaiver×2, Jetson TX2×2, Jetson Nano×1
S2	Jetson Xaiver×1, Jetson Nano×1
S3	Jetson Xaiver×1, Jetson TX2×1, Jetson Nano×1

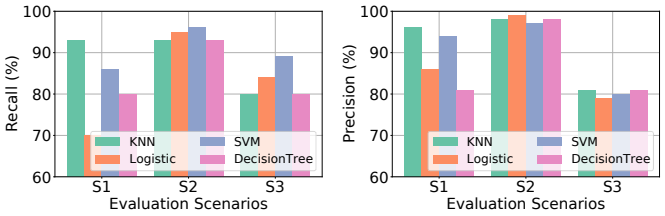
3) Object Detection Model: We use the YOLOv5[3] model implemented in PyTorch as the object detection network, with pretrained weight on COCO [19] dataset. We choose five sizes for partial frame detection: 64, 128, 256 and 512. Regions larger than 512 are downsampled to 512 as very large objects are easy to be detected. In the offline stage, we profile the YOLO inference time with 200 runs on each Jetson board and store the profiles as input to the BALB scheduling algorithm.

B. Cross-camera Correlation

We first compare our cross-camera object association algorithm with several baselines. We separately evaluate the two components contained in the association algorithm: the *classification module* and the *regression module*.

Classification Module: Given two cameras and an object bounding box in one camera, we want to know whether the same object appears in the other camera. If yes, the output is positive; otherwise, the output is negative. We compare the KNN model with three widely used binary classification baselines: support vector machine (SVM), logistic classification, and decision tree. We utilize *precision* and *recall* as the evaluation metrics. Results are presented in Figure 10.

[3]https://github.com/ultralytics/yolov5



(a) Classification recall. (b) Classification precision.

Fig. 10: Classification model comparisons.

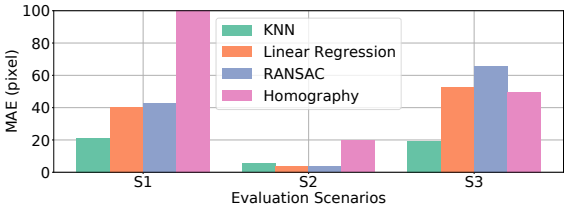


Fig. 11: Regression model comparisons.

Precision is more critical than recall in this case because a false positive means we falsely identify two distinct objects as one object and skips one of them. The results show the KNN model achieves better precision than the baselines, except in scenario S2, where logistical classification is slightly better. Precision and recall for scenario S3 are generally lower because of the limitation in labeling. S3 shows a busy traffic intersection scenario, but the released labels do not count partially occluded objects which may cause confusion to the classification model.

Regression Module: Given two cameras and the bounding box of an object in one camera, we want to infer its corresponding bounding box location in the other camera. We use *mean absolute error (MAE)* between bounding box coordinates as the metric for evaluating the regression models, and we choose the following three baselines.

- *Homography*: We use homography transformation [20] to map object bounding box locations between cameras.
- *Linear regression*: It can be regarded as a "learnable homography transformation" that learns a linear relationship between input and output bounding boxes.
- *Random sample consensus (RANSAC)* [21]: A robust regression model in the presence of many data outliers.

Figure 11 shows the comparison result for regression models. In scenario S1 and S3, KNN reaches the lowest MAE, while in scenario S2, it shows similar performance as linear regression and RANSAC. Homography performs much worse than KNN in all scenarios, because it can only map points in a 2D plane like ground in two cameras but not the bounding box coordinates, which can be affected by the object sizes (in all three dimensions including height) and facing directions.

C. Impact on Detection Quality

In this section, we evaluate the impact of the proposed framework on the resulted detection quality. Ideally, we want to optimize the neural network processing speed without missing any object appeared in the view.

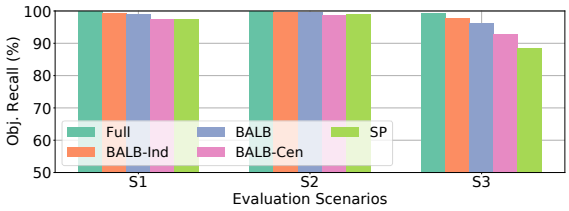


Fig. 12: Comparisons on object recalls for different algorithms.

Metric: We use *object recall* as the quality metric here. It is calculated as: At every timestamp, for each groundtruth object, as long as there is at least one camera detects it, then it is counted as a true positive. Otherwise, it is counted as a false negative. The object recall is defined as the ratio between true positives and all groundtruth objects. It is not affected by the missing labels for partially occluded objects.

Baselines: The following baselines are compared:

- *Full*: We perform full frame detection on every frame collected by every camera.
- *BALB-Ind*: Each camera independently runs the BALB framework without considering the spatial correlations among cameras.
- *BALB-Cen*: A variant of BALB that only runs the central stage algorithm to assign objects to cameras. No distributed stage is executed.
- *Static Partitioning (SP):* A fixed policy that partitions the overlap regions among cameras in offline according to their processing power. Each camera only tracks objects within its allocated region at regular frames.

Analysis: The results are summarized in Figure 12. Note the full-frame inspection results are used as the upper bound of recall for the remaining scheduling algorithms, which results in high inference time as we will show in Figure 13. We have the following observations: First, tracking-based slicing shows almost no degradation on detection recalls, as BALB-Ind achieves similar object recall as full frame detections in all scenarios. Second, through comparing BALB and BALB-Cen, we find that the central stage alone achieves high recall when only a few objects appear at a time (*i.e.*, S1 and S2). However, there is a degradation on BALB-Cen under complicated scenarios with busy traffic observed (*i.e.*, S3). This is where the distributed stage of BALB helps. Third, the imperfect object correlation model has a larger impact on SP than BALB-Cen in scenario S3, because we performed a matching step to associate detected objects and predicted object locations in BALB-Central stage, which reduces the false positives in the identified associations. In conclusion, the complete BALB algorithm provides better quality assurance than using each stage individually.

D. Impact on Processing Speed

We next evaluate the achieved speedup on the inference efficiency. We compare the average per-frame YOLO inference time on the slowest camera for each scheduling horizon. For BALB algorithms, we average the full frame inference time with regular frame times in a scheduling horizon. We

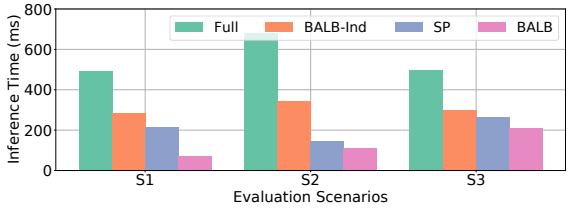


Fig. 13: Comparisons on per-frame YOLO inference latency.

TABLE II: Breakdown Per-frame Latency Overhead

Scenario	Central Stage	Tracking	Distributed BALB	Batching	Total
S01	2.59ms	18.90ms	0.08ms	7.53ms	29.10ms
S02	1.11ms	21.43ms	0.09ms	13.21ms	35.84ms
S03	2.27ms	11.55ms	0.22ms	19.86ms	33.90ms

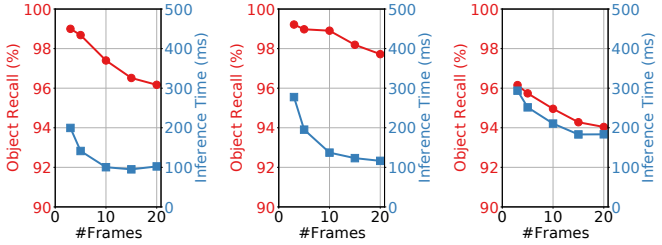
compare BALB with full frame inspections (Full), BALB-Ind, and Static Partitioning (SP) to progressively illustrate the advantages of BALB. The corresponding results are presented in Figure 13. We can see that BALB-Ind saves near 50% time compared to full frame inspections in all three scenarios by slicing and batching. Among the scenarios, the most time saving is observed in S2 while the least saving is observed in S3, because S2 has sparse vehicle distribution while S3 deals with busy traffic. Beyond that, BALB further saves 75%, 68%, 33% inference time compared to BALB-Ind in the three scenarios. The saving on scenario S3 is relatively small because the cross-camera view overlaps are smaller compared to the other two scenarios. Putting them together, we attain multiplicative speedups of $6.85\times$, $6.18\times$ and $2.45\times$ on BALB compared to full frame inspections in the three evaluation scenarios. The efficiency saving differs by scenarios depending on the number of appeared objects and camera view overlaps. Besides, BALB consistently outperforms SP because it not only considers the processing power disparity among cameras, but also the dynamics in camera latency skewness.

E. Impact of Scheduling Horizon Length

In this section, we investigate the impact of the scheduling horizon length on the attained object recall and frame neural network inference time. In Figure 14, we plot the change of object recall and inference time w.r.t the number of frames in a scheduling horizon. The observation is that longer scheduling horizons leads to higher inference speed, because the penalty of full frame detections are distributed among more frames, but they also suffer from lower recalls, which is caused by the inaccuracy of camera correlation models and the tracking algorithm. On the contrary, short scheduling horizons attain higher object recalls at the cost of higher inference time. Choosing the scheduling horizon $T = 10$ achieves a good tradeoff between the detection quality and inference efficiency.

F. System Overhead

Last, we report the system overhead produced by the components in our framework. The results are summarized in Table II. For each component, we first compute the maximum overhead among cameras at each frame, and then compute the



(a) S01 (b) S02 (c) S03

Fig. 14: The impact of scheduling horizon length on object recall and YOLO inference time.

mean overhead across frames. Since the central stage is only called once per scheduling horizon, we distribute its overhead to every frame. The central stage overhead includes both cross-camera object association and central BALB scheduler. The optical flow estimation is not blocked by any other steps, so we put it in a separate thread to run in parallel, such that its computation does affect the main thread efficiency. The resulted overhead per frame is between 29.10ms and 35.84ms. We can see the overhead mainly comes from the tracking and task batching, which may be further optimized by putting more engineering effort. We leave it as a future investigation.

V. Limitations and Discussion

In this section, we discuss a few limitations in our current system and their potential future solutions.

Imperfect object association: The cross-camera association is generally a challenging problem. Although our proposed data-driven approach outperforms the baselines in the evaluation, its precision is still not perfect, especially under busy traffic conditions, where occlusions and congestion happen frequently. The system should be further optimized for these situations. We may allocate multiple cameras to track the same object since we are not confident whether the two detections correspond to the same physical object.

Dynamic occlusion: Even if associations among objects viewed by different cameras were perfect, there are still reasons to associate multiple cameras with each object. One is the possibility of dynamic occlusions. For example, an object assigned exclusively to a camera might later get occluded by another object making it invisible to that camera, whereas it might remain visible to another camera. Taking future object trajectories into account (to predict future occlusions) can mitigate this effect. Assigning objects to multiple cameras with sufficiently different vantage points can also reduce occlusion-related failures. This effect needs further investigation.

Heterogeneity among cameras: We only consider the different processing capacity, but did not fully address other forms of heterogeneity among cameras (*e.g.*, frame rate and resolution), which may affect the tracking quality of the same target by different cameras. Besides, the observing distance and angle can also affect the tracking quality. One potential solution is to introduce a tracking quality metric in the

scheduling framework, such that the scheduling objective is extended to optimizing the quality-efficiency tradeoff, instead of purely minimizing the frame processing latency.

Object size: It is well known that objects closer to the camera are generally easier to classify [22]. Thus, all else being equal, assigning an object to a camera that is closer (*e.g.*, one where the objects accounts for more screen pixels) might help improve classification accuracy. The same policy will also tend to increase total load compared to a policy that is oblivious to objects' distance from the camera. The resulting trade-off between quality and resource savings must be explored.

Imperfect synchronization: Another limitation is that the approach requires the cameras to be approximately synchronized, so that they process the same scene at the same time. It might be, however, that the computational pipeline on one camera is slower. Thus, while some cameras are processing the "current" scene, others might still be working on older versions of the scene. In this case, anomalies may occur. For example, a camera might decide that an object has left its field of view and has entered the view of another camera. At the same time, the other camera might be "lagging" and so has not seen the object enter its field of view yet. Thus, both cameras might lose the current position of the object for some interval of time. This effect needs further investigation.

The assumption of view overlap: One important efficiency improvement in our system comes from the assumption of overlap in camera views. However, this assumption may not hold in more general multi-camera sensing applications, where the cameras might not share intersecting views. To extend the load balancing idea to those scenarios, we can consider allowing heavy-loaded cameras to offload some of their partial-frame inspections to the nearby idle cameras, but both the transmission delay and the computation latency need to be counted in the scheduling algorithm.

Extension to centralized processing: In some applications, the cameras are not equipped with GPUs so the videos can not be processed onboard. Instead, the images/videos should be offloaded to an edge/cloud server for centralized processing, where the network bandwidth becomes the resource bottleneck. The multi-view scheduling idea may be extended to this scenario by scheduling only one camera to upload its images or by uploading the minimum number of views that offers complete coverage of all objects. This is to save bandwidth consumption. The work also calls for innovations on data compression algorithms (*e.g.*, JPEG/MPEG).

Alternative problem formulations: Finally, the problem addressed in this paper was one of object assignment that minimizes the maximum object processing latency among a set of cameras. This formulation is motivated by the need to minimize response-time to any unusual conditions that need attention. Multiple alternative versions of the problem are possible and may be explored in future work. For example, in applications where real-time response is not needed, an alternative formulation might simply minimize the cumulative processed workload, possibly subject to some fairness constraints, or perhaps minimize consumption of a different resource, such as energy, as opposed to latency. Those and other formulations remain to be explored.

VI. Related Work

In this section, we briefly review the recent literature on video processing systems. Most early work [5], [23]–[25] on this topic focused on optimizing query-based video analytics. A query is provided and the system needs to automatically find all related information in a large-scale video database. They either optimize the indexing policy and storage structure for video data [23], or reduce the searching and querying effort by filtering out unrelated information [24].

Recent attention [3], [16], [26], [27] was paid to live video analytics systems, where the system needs to coordinate deployed cameras to perform query-based tasks, or general detection tasks, on the live video streams. These systems work in a traditional client-server architecture, where the cameras send the sampled image frames to the cloud for centralized processing. The key idea in this thread is to minimize the amount [3], [26], [28], resolutions [27], or the regions [16] of frames to be transmitted, because the network bandwidth is the main bottleneck in this pipeline.

With increasing compute power on smart cameras, one can provide a better real-time response if neural network processing was done locally. Existing work that utilizes camera compute power [29]–[32] focused on partitioning the workload between camera and edge/cloud servers. Glimpse [33] introduced the concept of intermittent cloud-based DNN execution, combined with local optical flow based tracking, to support continuous, real-time object recognition. This paper, to the best of our knowledge, is the first to optimize the neural network processing speed of a live video processing system purely at the edge devices, by exploiting the spatiotemporal correlations in the multi-view video streams and task batching mechanism on modern GPUs.

VII. Conclusion

We proposed a new paradigm of collaborative, *fine-grained* workload-sharing among a set of cameras under FoV overlaps, involving the *dynamic* and *autonomous* partitioning of individual object detection/tracking responsibilities among the set of collaborating cameras. It is achieved by fully exploiting the spatial-temporal data correlations in multi-view video streams, and effective task batching on GPU platforms. Through extensive evaluation using the NVIDIA AIC21 dataset on a heterogeneous edge testbed, we demonstrated the superiority of the proposed system in optimizing neural network processing speed, which achieves multiplicative speedups by $2.45\times$ to $6.85\times$ compared to processing full image frames, while the object tracking accuracy remains competitive.

Acknowledgement

Research reported in this paper was sponsored in part by DARPA award W911NF-17-C-0099, the Army Research Laboratory under Cooperative Agreement W911NF-17-20196,

NSF CNS 20-38817 and the National Research Foundation, Singapore under its NRF Investigatorship grant (NRF-NRFI05-2019-0007). The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies of the CCDC Army Research Laboratory, DARPA, the US government or the National Research Foundation, Singapore. The US government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, "Cityflow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, p. 8797–8806.

[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)*, ser. NSDI '17, 2017.

[3] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 359–376.

[4] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.

[5] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.

[6] 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.

[7] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019, pp. 9–14.

[8] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2020, pp. 319–332.

[9] S. Liu, S. Yao, X. Fu, H. Shao, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "Real-time task scheduling for machine perception in intelligent cyber-physical systems," *IEEE Transactions on Computers*, 2021.

[10] 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.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] S. Liu, X. Fu, M. Wigness, P. David, S. Yao, L. Sha, and T. Abdelzaher, "Self-cueing real-time attention scheduling in criticality-driven visual machine perception," in *2022 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022.

[13] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *2008 IEEE Conference on computer vision and pattern recognition*. IEEE, 2008, pp. 1–8.

[14] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[15] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast optical flow using dense inverse search," in *European Conference on Computer Vision*. Springer, 2016, pp. 471–488.

[16] 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.

[17] S. He, H. Luo, P. Wang, F. Wang, H. Li, and W. Jiang, "Transreid: Transformer-based object re-identification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 013–15 022.

[18] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

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

[21] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[22] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021.

[23] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 269–286.

[24] Y. Zhang and A. Kumar, "Panorama: a data system for unbounded vocabulary querying over video," *Proceedings of the VLDB Endowment*, vol. 13, no. 4, pp. 477–491, 2019.

[25] M. R. Anderson, M. Cafarella, G. Ros, and T. F. Wenisch, "Physical representation-based predicate optimization for a visual analytics database," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1466–1477.

[26] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

[27] 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.

[28] S. Paul, U. Drolia, Y. C. Hu, and S. T. Chakradhar, "Aqua: Analytical quality assessment for optimizing video analytics systems," *arXiv preprint arXiv:2101.09752*, 2021.

[29] 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.

[30] 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.

[31] 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.

[32] 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 International Conference on Embedded Networked Sensor Systems (SenSys)*, 2020.

[33] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015, pp. 155–168.