Self-Cueing Real-Time Attention Scheduling in Criticality-Aware Visual Machine Perception

Shengzhong Liu[†], Xinzhe Fu[‡], Maggie Wigness[§], Philip David[§], Shuochao Yao[*], Lui Sha[†], Tarek Abdelzaher[†]
[†]University of Illinois at Urbana-Champaign, [‡]Massachusetts Institute of Technology,
[§]US Army Research Labs, [*]George Mason University
Email: sl29@illinois.edu, xinzhe@mit.edu, {maggie.b.wigness.civ, philip.j.david4.civ}@mail.mil,
shuochao@gmu.edu, {lrs, zaher}@illinois.edu

2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-9998-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/RTAS54340.2022.00022

***Abstract*—This paper presents a self-cueing real-time framework for attention prioritization in AI-enabled visual perception systems that minimizes a notion of state uncertainty. By *attention prioritization* we refer to inspecting *some parts of the scene before others* in a criticality-aware fashion. By *self-cueing*, we refer to not needing external cueing sensors for prioritizing attention, thereby simplifying design. We show that attention prioritization saves resources, thus enabling more efficient and responsive real-time object tracking on *resource-limited embedded platforms*. The system consists of two components: First, an optical flow-based module decides on the regions to be viewed on a subframe level, as well as their criticality. Second, a novel batched proportional balancing (BPB) scheduling policy decides how to schedule these regions for inspection by a deep neural network (DNN), and how to parallelize execution on the GPU. We implement the system on an NVIDIA Jetson Xavier platform, and empirically demonstrate the superiority of the proposed architecture through an extensive evaluation using a real-word driving dataset.**

I. Introduction

Attention prioritization and scheduling is a novel problem in real-time systems literature [1] that refers to algorithms for prioritizing and scheduling of data processing at a *subframe level* in data intensive workflows, such as neural-network-based camera or LiDAR processing. The problem is cyber-physical in nature in that our physical understanding of the importance of observed objects in various parts of the scene drives the needed fidelity of their real-time tracking and thus the frequency/priority at which they need to be inspected and localized by the AI in the loop. This is as opposed to applying the AI to entire (video or LiDAR) frames in a FIFO manner.

This paper introduces a self-cueing attention prioritization and scheduling framework for visual machine perception pipelines in cyber-physical systems that minimizes a notion of uncertainty in object location. Modern machine perception pipelines rely on neural networks (*e.g.*, YOLO [2]), to perform object detection, localization and classification tasks (thereafter collectively called *detection tasks* for short, where no ambiguity arises), and feed downstream components such as navigation control. Attention prioritization reduces the area inspected by the detection neural network in some criticality-dependent fashion to improve perception efficiency. It mimics the allocation of human cognitive capacity to focus on elements that matter most in a complex scene, as opposed to giving all elements of the scene the same level of attention. Unlike previous work that relied on an external ranging sensor to cue attention [1], in this paper, we do it without external cueing.

Interestingly, removing the external cueing sensor fundamentally changes the attention scheduling problem. External cuing (*e.g.*, based on LiDAR distance) yields, *for every frame*, an estimate of locations or regions of interest to be inspected by the AI, thereby reducing the amount of processing needed. In contrast, in the absence of such an external cue, a full frame needs to be processed occasionally to detect all objects of interest first. Uncertainty in object locations then keeps growing with duration elapsed since the last time a full frame was processed. To bound the growing uncertainty in each object's location, inspection of more uncertain regions (*e.g.*, of faster or more erratically-moving objects) must be scheduled more often. Thus, scheduling is coupled to uncertainty growth; smaller inspection periods reduce accumulated uncertainty and vice versa (where the rate of uncertainty accumulation depends on the object). This coupling between scheduling policy and uncertainty accumulation levels is different than in the case of an external cueing sensor, in that the sensor decouples the two – uncertainty does not accumulate but rather becomes a function of the cueing sensor only (that is always on). The problem of attention scheduling to bound uncertainty accumulation does not arise.

We formulate our scheduling problem as one of minimizing maximum weighted (location) uncertainty, and develop a near-optimal real-time scheduling algorithm to schedule the selected regions for processing (by the perception neural network) on the GPU. Autonomous driving is used as an example application, although the design generalizes to other cyber-physical applications as well, such as delivery drones and surveillance applications.

The work is motivated by the increasing popularity of *visual machine perception* (*i.e.*, the process of extracting relevant knowledge of immediate surroundings from camera images) as the foundation for many intelligent cyber-physical applications [3]–[5]. Advances in deep neural networks have significantly improved the perception quality of many vision tasks, such as image recognition [6], object detection [7], [8], and semantic segmentation [9]. However, delivering real-time results by running computationally intensive neural network

978-1-6654-9998-9/22/$31.00 ©2022 IEEE
DOI 10.1109/RTAS54340.2022.00022

models on resource-limited embedded platforms has remained a key challenge that significantly increases the cost of autonomy. Additionally, as pointed out by Huang *et al.* [10], objects in driving scenarios are about three times smaller on the image plane than objects in general detection scenarios. Thus, downsizing image resolutions to improve the inference efficiency is an unreliable solution with respect to the detection quality in autonomous driving.

Many existing efforts on enabling real-time neural network inference on embedded platforms focus on neural network compression [11], [12] and cloud offloading [13], [14]. There are a couple of limitations with these approaches. On one hand, in compression, detection quality of all parts of the scene is affected with no regard to criticality. On the other hand, offloading (part of) the computation to the cloud requires a stable and fast network connection, which is not guaranteed in many critical application scenarios. As an alternative approach, Liu *et al.* [1] proposed to slice input images into regions of different criticality, so that inspection of critical regions can be prioritized. The work relied on an external ranging sensor (*e.g.*, LiDAR) to determine which parts of the scene are more critical to inspect first. Unfortunately, LiDARs are expensive, have a limited range, and are not available on all platforms (*e.g.*, Tesla has famously opposed using LiDAR). Furthermore, reliance on multiple sensors increases cost and requires precise calibration and synchronization, where the degradation of either could cause downstream detection issues. By introducing a self-cueing attention scheduling framework that works without dependency on external sensor inputs, we side-step the above fusion challenges.

We implement the proposed framework on NVIDIA Jetson Xavier, and empirically evaluate its performance using real world driving datasets. The results show that the proposed policy achieves high detection, localization, and classification quality (compared to baselines) under different workloads. It also provides better responses to physically close objects.

The rest of this paper is organized as follows: In Section II, we briefly review the related literature. We give an overview of the architecture in Section III, then explain the data slicing module in Section IV and the scheduling algorithm in Section V. We discuss some adopted empirical optimizations in Section VI, before presenting the evaluation results in Section VII. Finally, we conclude the paper in Section VIII.

II. Related Work

A. Real-time Machine Perception

Most previous research to support real-time machine perception focused on compressing neural networks to reduce the inference latency [11], [12], [15]–[17]. However, existing compression approaches do not offer the flexibility to tailor the degree of compression at a subframe level. Recently, real-time scheduling has emerged as a key challenge in AI-based perception systems [18]. Related work can be divided into three categories: (i) system-level scheduling; (ii) model-level scheduling; and (iii) data-level scheduling. System-level scheduling algorithms try to optimize CPU-GPU interactions by appropriately allocating and pipelining the computational stages [19]–[22]. In contrast, model-level scheduling algorithms dynamically adjust the utilized neural network structures to meet inference deadlines [23]–[26]. Finally, the data-level scheduling algorithms slice the data into partial regions, and process them at a fine-grained and criticality-aware manner. One drawback of existing approaches [1], [27], [28] lies in their reliance on an external attention cueing sensor (*e.g.*, a ranging LiDAR), which may be not be an option in some autonomous systems. In this paper, we build a self-cueing system that only relies on the original data flow without external secondary sensor cues.

B. Temporal Correlations in Video Object Detection

Our self-cueing scheme fundamentally relies on objects permanence to hypothesize that object observed in earlier frames will still be located some bounded distance away in the current frame. In other words, frames are highly correlated. Video temporal correlations have been extensively studied in continuous object detection. Some papers, including [29]–[32], rely on motion vectors between consecutive frames to reduce the network depth to extract features on new frames. They utilized motion vectors to map (part of) past features into the new frame. Buckler *et al.* [33] proposed an optical flow-based hardware solution to propagate latent features from previous frames to the new frame. The uncertainty in the estimated motions is not counted. Some work also leverages pairwise image differences to guide an object detector to focus on changing areas in the new frame [34], [35]. However, they are only applicable to statically mounted cameras, which no longer works in autonomous driving systems. Song *et al.* [36] applied different quantization levels to process regions with different sensitivity on the same frame, which was only limited to image classification models. Both Kumar *et al.* [37] and Mao *et al.* [38] proposed to use object tracker projections to extract regions of interest in the new frame. We build on such prior solutions, using them to determine possible object locations in the current frame, ahead of actual frame inspection by the (AI-based) perception subsystem, thus providing an input into our attention prioritization and scheduling problem.

III. System Overview

Assume the system uses a camera to continuously observe its surroundings at a fixed frame rate. An object detector (e.g., YOLO) is used to localize and categorize all objects in the captured image frames. The detector can accept variable image sizes as input and has an inference latency that depends on input size. The deployed detector is computationally intensive such that inspection of a full image can not finish before the next frame arrives. Instead, we inspect full frames at a longer interval T (say, 1-2 seconds). We refer to processing of full frames as *full-frame inspections*. Between them, we identify regions of interest using optical flow [39], a much faster algorithm (than neural networks) that compares successive frames and estimates approximate motion vectors for pixels. It is used to guess (within some error margin) where objects

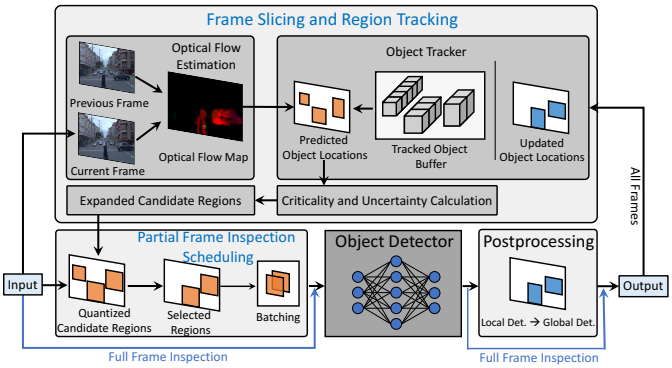


Fig. 1. The overview of the proposed scheduling framework. It is generally applicable to any object detector.

of interest, detected in previous frames, might have moved to in the current one. The attention scheduler then decides which of these regions are to be inspected by the AI-based perception subsystem (the detector) for exact localization of the corresponding objects. We call such selective processing *partial-frame inspections*. We define the time between two full-frame inspections as a *scheduling horizon*, and propose a novel scheduling algorithm to decide the schedule of partial-frame inspections within each horizon to minimize the *maximum weighted location uncertainty*.

Two core components are included in the proposed architecture: (i) the *frame slicing and region tracking module*, and (ii) the *partial-frame inspection scheduling module*. An overview of the architecture is shown in Figure 1.

Frame Slicing and Region Tracking: This module slices image frames (between full-frame inspections) into regions where objects may be present. After a full-frame inspection localizes all objects in a frame, an optical-flow based tracking algorithm tracks the object locations in subsequent frames (until the next full-frame inspection). Within each frame, the module determines the approximate regions that contain these tracked objects, taking into account the uncertainty in their predicted locations. These regions are the candidates to be inspected by the detector. Background regions are filtered out.

Partial-Frame Inspection Scheduling: This module selectively schedules the appropriate (partial frame) regions for inspection by the detector, in order to maintain bounded low uncertainty in the location each tracked object. Every object is associated with a criticality indicating its application-level importance, and an uncertainty growth rate. Object uncertainty increases monotonically (at possibly different rates) with time if no new inspection is performed. Partial-frame inspections of objects with low criticality or slow uncertainty growth are scheduled less frequently. In other words, the inspection of regions containing these objects is skipped in some frames. In addition, when making decisions, the scheduling algorithm considers *task batching* on modern GPUs, which means multiple regions can be batched together and submitted as a single GPU request, *as long as they have the same size* (because low-end GPUs can only batch identical computational kernels). Batched processing can achieve much lower latency than serialized processing.

IV. Frame Slicing and Region Tracking

In this section, we introduce the optical flow-based tracking algorithm, and explain how it induces location uncertainties.

A. Optical Flow Background

Optical flow algorithms take two consecutive frames as input and estimate the pixel-level motion vectors between them, as caused by the relative movement between objects and the observer. They return a map of single pixel motions, which is called *optical flow map*. The RGB image is first converted to gray scale, where each pixel value represents the light intensity at that location. We use $I(x, y, t)$ to denote the image intensity at pixel (x, y) of frame t. The optical flow map is a matrix of coordinate displacements (d_x, d_y), such that,

$$I(x, y, t) = I(x + d_x, y + d_y, t + 1). \tag{1}$$

Optical flow assumes that the pixel intensities of an object are constant across two consecutive frames. In this paper, we use the DIS method [39], which is a widely used and efficient optical flow estimation algorithm.

In autonomous driving, although the relative movement between the camera, object and light sources may cause drastic change on the lighting and reflection on objects (especially during night driving), optical flow tends to err on the safe side. Specifically, it may introduce false positives (*e.g.*, spurious areas may be highlighted for visual inspection), but is less likely to produce false negatives (*i.e.*, missing actual changes in locations of visible objects). Merging the trajectory-based tracking (or a physical mobility model) with optical flow observations can indeed reduce false positives (similar to the way Kalman filters leverage both imperfect models and imperfect empirical observations to produce improved trajectory estimates), and we leave it as one of our future directions.

B. Optical Flow-based Object Tracking

The motivation of using optical flow for tracking primarily comes from its high reliability in identifying an appropriately expanded region around the predicted object location for further DNN inspection without experiencing an model converging period. We use optical flow as a non-parametric motion model to estimate possible object locations in intermediate frames based on *observed pixel movement*. We chose it over conventional parametric tracking models, such as Kalman Filters (*e.g.*, in SORT [40]), because the latter models often require a sequence of past observations to correctly estimate trajectories, which might fail to correctly predict location in the presence of sudden unexpected movement (such as swerves to avoid an obstacle). Instead, as discussed above, the sensitive property of optical flow makes it less likely to miss actual object movement and achieve higher recall on retaining tracking.

Algorithm 1 details our tracking algorithm. We start from the set of objects detected by the last full-frame inspection. Each time a new frame arrives, we first compute its optical

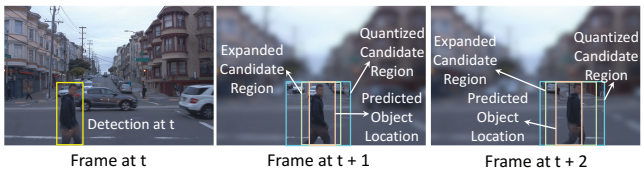


Fig. 2. Comparison between different regions used in the tracking.

Algorithm 1: Optical Flow-based Object Tracking

Input: Set of object $\{1, \ldots, N\}$, $K-1$ frames between two full-frame inspections, object detector.

Maintain a set of object tracks for target objects;
for *frame* $k = 2, \ldots, K$ **do**
  Calculate the flow map between frame k and $k-1$;
  for *object* $i = 1, \ldots, N$ **do**
    Calculate object representative flow $(dx_i^{(k)}, dy_i^{(k)})$ by taking the median flow of previous object location;
    Update tracked object center location $\tilde{cx} := cx_i^{(k-1)} + dx_i^{(k)}$, $\tilde{cy} := cy_i^{(k-1)} + dy_i^{(k)}$;
  end
  Generate set of partial detections by the object detector;
  Data association using Hungarian algorithm between object tracks and new detections using IoU metric;
  for *object* $i = 1, \ldots, N$ **do**
    if *mapped with a new detection* **then**
      new object location := mapped detection location
    end
    else
      new object location := predicted object location
    end
  end
end

flow map compared to its preceding frame, then calculate the following three regions for each tracked object:

① **Predicted Object Location:** It tightly bounds the most likely (predicted) object location from the optical flow map. We use this updated location as a best guess of current object location in the absence of an actual object inspection.

② **Expanded Candidate Region:** It expands the predicted location on account of uncertainty. This is the area that should be inspected by the detector if we want to localize the object again. It is a box whose area keeps expanding until an inspection of this region is scheduled.

③ **Quantized Candidate Region:** We pad the expanded candidate region to the nearest quantized size from a preset set. This is done to improve subsequent batching opportunities, since same-size images can be processed in parallel (batched) as will be discussed in more detail in Section V.

Figure 2 illustrates the difference between the three regions. Next, we explain how they are calculated.

1) Computing Predicted Object Locations: To compute the predicted location for an object, we compute the median motion vector of all pixels within the previous object bounding box, and move the bounding box by that vector. The median motion is chosen over the mean motion to eliminate the impact of outliers and static background pixels (*e.g.*, road or sky).

2) Computing Expanded Candidate Regions: This region starts from a previously detected object location, and then

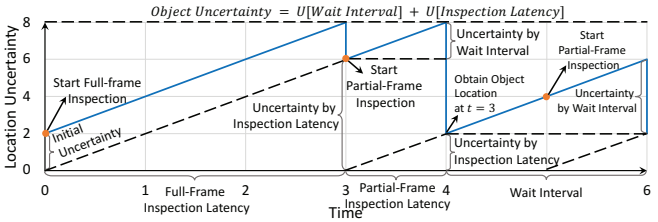


Fig. 3. The object uncertainty comes from both *wait intervals* and *inspection latencies*. The reset value only depends on inspection latency.

keeps expanding, until a new detection is made. Specifically, at a new frame k, we use $[x_{min}^{(k)}, y_{min}^{(k)}, x_{max}^{(k)}, y_{max}^{(k)}]$ to denote the new object location, and use $\mathbf{D} = [\mathbf{D}_{\hat{x}}, \mathbf{D}_{\hat{y}}]$ to denote the partial flow matrix corresponding to its previous expanded candidate region, say $[\hat{x}_{min}^{(k-1)}, \hat{y}_{min}^{(k-1)}, \hat{x}_{max}^{(k-1)}, \hat{y}_{max}^{(k-1)}]$. If the previous expanded candidate region completely covers the previous object location, then the new object location satisfies:

$$\hat{x}_{min}^{(k-1)} + \min_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}} \le x_{min}^{(k)} \le x_{max}^{(k)} \le \hat{x}_{max}^{(k-1)} + \max_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}},$$
$$\hat{y}_{min}^{(k-1)} + \min_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}} \le y_{min}^{(k)} \le y_{max}^{(k)} \le \hat{y}_{max}^{(k-1)} + \max_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}}.$$

Thus, we define the new expanded candidate region as:

$$[\hat{x}_{min}^{(k-1)} + \min_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}},\ \hat{y}_{min}^{(k-1)} + \min_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}},$$
$$\hat{x}_{max}^{(k-1)} + \max_{d_{\hat{x}} \in \mathbf{D}_{\hat{x}}} d_{\hat{x}},\ \hat{y}_{max}^{(k-1)} + \max_{d_{\hat{y}} \in \mathbf{D}_{\hat{y}}} d_{\hat{y}}].$$

The expansion considers the possibly different pixel motions for different parts of an object. Since the expanded candidate region starts from the exact object location, it holds by induction that the expanded candidate region will cover the (groundtruth) object location at every future frame, if the estimated optical flows are accurate.

3) Computing Quantized Candidate Regions: To facilitate batching of image processing, we pad the expanded candidate region to the nearest quantized *target size* s_i chosen from a finite set, $s_i \in \{s_1, \ldots, s_M\}$. The padded region is then called the *quantized candidate region*. We assign a fixed padded target size s_i to each object within a scheduling horizon. We provide two justifications for this choice: First, the quantized size is larger than the initial object size, so it leaves space for object size increase in upcoming frames. Second, if the expanded candidate region increases beyond s_i, we reduce its resolution to make it fit into s_i, because downsizing large objects does not degrade their perception quality [41].

4) Data Association: After we receive the detected object locations from the detector, we perform data associations between the existing object tracks (represented by their predicted object locations) and the newly detected bounding boxes. We do so by using the Hungarian algorithm based on their location overlaps with an Intersection-over-Union (IoU) metric. We then update the mapped object locations to the newly detected locations. Those objects not inspected by the detector in a given frame will retain their predicted object locations.

C. Object Location Uncertainty

The *object location uncertainty* reflects our confidence on the predicted object location. Intuitively, if the size of the expanded candidate region is close to the predicted object location, we have a low uncertainty (*i.e.*, high confidence) on the predicted object location; otherwise, if the object can appear at much larger area than the predicted object location, we have a high uncertainty (*i.e.*, low confidence) on the predicted object location.

We assign an *object weight* $w_i = v_i \cdot u_i$ to each tracked object $\mathcal{O}_i$, where: (1) v_i is the *object criticality* representing the application-specific importance, which is static within a scheduling horizon. This is a policy decision that is outside the scope of this paper. Even though the visual images do not directly contain distance information, it is still possible to distinguish the nearby and distant objects by combining object locations with background (*i.e.*, the road), which we assume is separately solved by other AI techniques. (2) u_i is the *uncertainty growth rate*, defined as the average rate of its candidate region expansion. After we obtain the full-frame inspection result, we calculate the uncertainty growth rate as $u_i = \sqrt{S_i^{ECR}/S_i^D}/t_f$, where S^{ECR} is the area of the expanded candidate region, S_i^D is the area of the detected location, and t_f is the latency of full-frame inspection. The uncertainty grows linearly with time if no new inspection is performed, which relies on the assumption that we have a sufficiently small gap between consecutive inspections, such that the concatenation of linear segments can closely approximate the actual uncertainty growth.

As shown in Figure 3, the overall object uncertainty comes from two sources: *wait intervals* and *inspection latencies*. Wait interval is defined as the elapsed time since we obtained the last inspection result. Inspection latency refers to the time running the last inspection task. The second part exists because the obtained object location does not correspond to the finish time of the inspection task, but its start time. After each inspection task, the uncertainty is reset to the value solely caused by the inspection latency. By separating the uncertainty into the weight factor and the elapsed time, we can simply denote the weighted uncertainty of object $\mathcal{O}_i$ as $U_i(t) = w_i(t - t_i) + u_i$, where $t - t_i$ is the elapsed time since the end of its last inspection, and u_i is the uncertainty resulted from its inspection latency.

V. Partial-Frame Inspection Scheduling

In this section, we formulate the partial-frame inspection scheduling problem, and introduce the proposed algorithm.

A. Task Execution Model

We divide the time into fixed-length segments, where each segment is called a *scheduling horizon* T. K frames are captured within each scheduling horizon. The platform is equipped with a single GPU that runs the detector. We run a full-frame inspection at the first frame of each scheduling horizon, which identifies N objects $\{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_N\}$. We

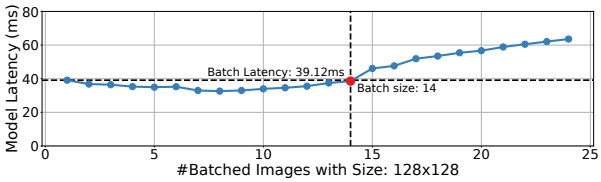


Fig. 4. YOLO execution latencies of 128×128 images with different batch sizes on Jetson Xavier. The inflection point is highlighted in red, where the batch size is 14. We set the batch limit and batch latency correspondingly.

subtract the latency of the preprocessing steps and the full-frame inspection to get the time budget for partial-frame inspections. Each object $\mathcal{O}_i$ is associated with a target size $s_i \in \{s_1, \ldots, s_M\}$, within horizon T, which restricts the size of its quantized candidate regions and facilitates batching. For each target size s, there exists a maximum number of regions that can be batched and processed in parallel on the GPU. We call it the *batching limit* κ_s for size s. Although the detector execution time can increase with the number of batched regions, by appropriately setting the batching limit, we operate in a region where execution time changes only slightly with batching (before an inflection point is reached where the slope increases, as shown in Figure 4). We denote the worst-case batch execution time by τ_s. In other words, the GPU can simultaneously run partial-frame inspections for κ ($1 \le \kappa \le \kappa_s$) objects of target size s within time τ_s.

B. Scheduling Problem Formulation

A good perception system should selectively run partial-frame inspections to maintain low location uncertainty on each object throughout the scheduling horizon. Recall that the (weighted) location uncertainty of object $\mathcal{O}_i$ at time t is $U_i(t) = w_i(t - t_i) + u_i$. Without loss of generality, we assume $w_1 \le \ldots \le w_N$. The maximum uncertainty for object $\mathcal{O}_i$ over the scheduling horizon is denoted by $U_i = \max_{t \in [0,T]} U_i(t)$. Our goal is to minimize the maximum weighted uncertainty over all objects, which we refer to as the *system uncertainty* U. It is defined as $U = \max_{i \in \{1,\ldots,N\}} U_i$. The problem we study, is to design a *schedule* of partial-frame inspections such that the system uncertainty is minimized. A schedule specifies the ordering and batching of partial-frame inspections.

Definition 1 (Schedule)**.** *A schedule is a sequence of tuples* $(\mathcal{N}^1, s^1, t^1, k^1), (\mathcal{N}^2, s^2, t^2, k^2), \ldots, (\mathcal{N}^I, s^I, t^I, k^I)$. *Both* $t^1, \ldots, t^I$ *and* $k^1, \ldots, k^I$ *are in non-decreasing order. For a generic* j*-th tuple, it represents the* j*-th batch, where:*

- $\mathcal{N}^j$ *is the subset of objects that get inspected in the batch. No object can appear more than once in the subset.*
- s^j *denotes the target size of the batch.*
- $t^j \in [t_f, T]$ *is the start execution time of the batch.*
- $k^j \in \{2, \ldots, K\}$ *represents the frame on which the partial-frame inspection is run.*

A schedule is *feasible* if it satisfies for each batch j: (1) The number of batched regions is within the batching limit, *i.e.*, $|\mathcal{N}_j| \le \kappa_{s^j}$. (2) We define the *valid period* of a frame as the interval between its arrival and the arrival of the next frame.

Algorithm 2: The BPB Policy

Input: Object set $\{\mathcal{O}_1, \ldots, \mathcal{O}_N\}$, weights $\{w_1, \ldots, w_N\}$, number of frames $K-1$ for partial-frame inspections.
Output: A feasible schedule with minimized uncertainty.

Sort and reindex the objects such that $w_1 \leq \ldots \leq w_N$;
for $i = 1, \ldots, N$ **do**
  $x_i := 2^{\lfloor \log_2(w_i/w_1) \rfloor}$;
end
$\mathcal{C} = \{\frac{1}{x_N}, \frac{1}{x_{N-1}}, \ldots, 1, 2, 3, \ldots, \lfloor \frac{K-1}{x_N} \rfloor, \frac{K-1}{x_N}\}$;
Binary search for the maximum $c \in \mathcal{C}$ such that the schedule computed by **Algorithm 3** for task set $\{\lfloor cx_1 \rfloor, \ldots, \lfloor cx_N \rfloor\}$ is feasible (*i.e.*, the finishing time is no larger than T);
Return the schedule for the task set of the maximum c.

Any batch can only run on the currently valid frame. (3) The start time of the batch is no earlier than the finish time of its previous batch, *i.e.*, $t^j \geq t^{j-1} + \tau_{s^{j-1}}$. (4) The finish time of the last batch is no later than T, *i.e.*, $t^I + \tau_{s^I} \leq T$.

Note that each feasible schedule can be executed on the physical machine, and each execution on the physical machine can be translated to a feasible schedule. With the above preliminaries, we formulate our problem as follows.

Definition 2 (Partial-Frame Inspection Scheduling Problem)**.** *The Partial-Frame Inspection Scheduling (PFIS) problem asks for a feasible schedule that minimizes the system uncertainty within a scheduling horizon.*

The PFIS problem requires us to carefully select subsets of objects to run and batch on each frame. Although it can be optimally solved by the dynamic-programming paradigm, the resulted computational complexity would be high. Instead, we will propose a low-complexity policy, called the Batched Proportional Balancing (BPB) policy, that computes approximately optimal schedules with provable uncertainty guarantee.

C. Scheduling Policy

The general idea of the proposed **Batched Proportional Balancing (BPB)** policy is to set the number of partial-frame inspection tasks for each object proportional to its object weight, such that the objects with high criticality or high uncertainty growth would receive more attention. For object $\mathcal{O}_i$, we use the *inspection frequency* x_i to denote its number of scheduled partial-frame inspection tasks within the scheduling horizon. The *inspection frequency set*, is thus defined as:

Definition 3 (Inspection Frequency Set)**.** *The inspection frequency set $\{x_1, \ldots, x_N\}$ is a set of inspection frequencies corresponding to the number of partial-frame inspection tasks of all objects in the scheduling horizon where, for each object $\mathcal{O}_i$, x_i partial-frame inspection tasks are scheduled.*

We aim at computing an inspection frequency test where the inspection frequency x_i for object $\mathcal{O}_i$ is *approximately proportional* to its weight w_i (*i.e.*, `Proportional`), and make sure the intervals between consecutive partial-frame inspections of each object are evenly distributed in the schedule (i.e. `Balancing`) The design so far seems similar to the well-studied pinwheel scheduling problem [42]. However, we go a step further by considering **task batching** (*i.e.*, `Batched`), where we need to simultaneously decide *when to detect each object* and *how to batch the inspections of objects* such that the system uncertainty is minimized. Improper batching may result in low utilization on the GPU and much higher system uncertainty. The pseudocode of the BPB policy is presented in **Algorithm 2**. It searches for an inspection frequency set with the minimum system uncertainty, and invokes the **Batch-Aware Scheduling (BAS)** algorithm (**Algorithm 3**) as a sub-procedure to derive an optimal schedule for a given inspection frequency set.

Algorithm 3: Batching-Aware Scheduling (BAS)

Input: Inspection frequency set $\{x_1, \ldots, x_N\}$
Output: A schedule for the inspection frequency set

```
// (1) Calculate the task-bin mapping.
```
$L := x_N$;
for $i \in \{N, N-1, \ldots, 1\}$ *(decreasing order of x_i)* **do**
  Let L_i be the first L/x_i bins $\{B_1, \ldots, B_{L/x_i}\}$;
  $s_i :=$ the target size of $\mathcal{O}_i$;
  if $\exists B_l \in L_i$ *with incomplete batch of size* s_i **then**
    Add the first task of $\mathcal{O}_i$ to B_l;
  end
  else
    Add the first task of $\mathcal{O}_i$ to the bin in L_i with the minimum load;
  end
  Replicate the mapping of the remaining tasks of n to the remaining subset of bins;
end

```
// (2) Convert the task-bin mapping to a schedule.
```
$j = 1, t^j = 0$, schedule $\mathcal{S} = \emptyset$;
for $l \in \{1, \ldots, L\}$ **do**
  $t_j := \max\{t_j,$ start of valid period of the l-th frame$\}$.
  for $s \in \{s_1, \ldots, s_M\}$ **do**
    $\kappa :=$ the number of objects of size s in B_l;
    while $\kappa > 0$ **do**
      $\mathcal{N}^j := \min\{\kappa, \kappa_s\}$ objects of size s in B_l;
      $k :=$ the most recent camera frame at t_j
      Add $(\mathcal{N}^j, s, t^j, k)$ to $\mathcal{S}$;
      $t^{j+1} := t^j + \tau_s$, $j := j + 1$;
      Remove the selected objects from B_l;
    end
  end
end
Return the schedule $\mathcal{S}$

To reduce the search effort, the BPB policy first proportionally derives the *normalized inspection frequencies* of objects such that the object with the smallest weight is detected only once. They are computed by dividing the object weights by the minimum weight, and rounding down to the nearest power of 2 if they are not[1]. Let the normalized inspection frequency set be $\{x_1, \ldots, x_N\}$. BPB then searches a maximum scaling factor c such that the schedule returned by the BAS algorithm for the inspection frequency set $\{\lfloor cx_1 \rfloor, \ldots, \lfloor cx_N \rfloor\}$ is feasible. Note that the scaling factor c can be smaller than one, and thus in the resulting inspection frequency set, $\lfloor cx_n \rfloor$ can be zero for some objects. Such objects will not be scheduled. As

[1]This operation is used to align the inspection times among objects to trigger more batching opportunities.

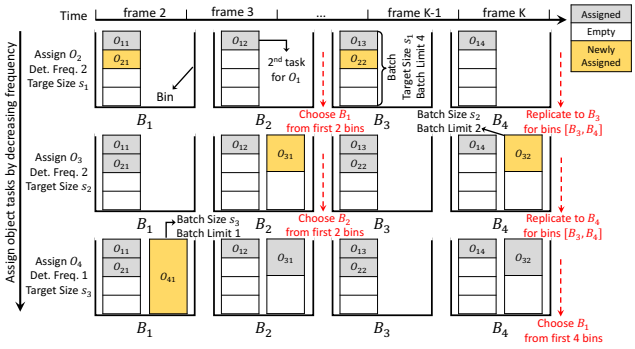


Fig. 5. Graphical illustration on how BAS generates the task-bin mapping. We have four objects denoted by (object, inspection frequency, target size): $(\mathcal{O}_1, 4, s_1)$, $(\mathcal{O}_2, 2, s_1)$, $(\mathcal{O}_3, 2, s_2)$, $(\mathcal{O}_4, 1, s_3)$. We have 4 (virtual) bins, which are **not** aligned with the frame boundaries. For object $\mathcal{O}_2$, its first task is assigned to bin B_1 because there is an incomplete batch with size s_1, and the decision is replicated to bin B_3. For object $\mathcal{O}_3$, its first task is assigned to bin B_2, and the decision is replicated to bin B_4. The task for object $\mathcal{O}_4$ is assigned to bin B_1 with the min load.

we will show in the sequel, if the schedule calculated by the BAS algorithm for c is feasible, so is the schedule calculated by the BAS for any $c' \leq c$. Thus, the maximum c can be identified via binary search due to this monotonicity property.

The Batch-Aware Scheduling (BAS) algorithm (**Algorithm 3**) computes an optimal schedule that minimizes the system uncertainty for a given inspection frequency set $\{x_1, \ldots, x_N\}$. BAS works as a two-step procedure. First, BAS maps the partial-frame inspection tasks for objects to $L = x_N$ temporally distributed virtual bins $\{B_1, \ldots, B_L\}$. The virtual bins do not correspond to camera frames. No object can have more than $K-1$ partial-frame inspections in a scheduling horizon, so we assume $L \leq K-1$. BAS sequentially assigns the tasks of each object $\mathcal{O}_i$ in decreasing order of x_i. Since each x_i is an integer power of 2 multiple of the minimum non-zero element in $\mathcal{C}$, when mapping tasks for object $\mathcal{O}_i$, BAS only designates the mapping of its first task to the first L/x_i bins[2] and replicates the mapping for the remaining tasks to the corresponding bins in remaining subsets. By doing so, when assigning tasks of an object, the matched bins in different subsets always have perfectly symmetric load. The first task of each object is assigned in a batch-aware load-balanced fashion. At object $\mathcal{O}_i$, BAS first checks whether there is a bin that has incomplete batch with size s_i , *i.e.*, the number of tasks with size s_i in the bin is not a multiple of κ_{s_i}. If such a bin exists, it assigns the task to that bin; otherwise, it assigns the task to the bin with the minimum load. The *bin load* λ_l is the execution time sum for batches in bin B_l. The assignment process is visually illustrated in Figure 5. Second, it converts the generated task-bin mapping to a schedule by sequentially executing the bins, and greedily batching tasks with the same target size in each bin. When compositing a batch, we select the valid frame at that time to run partial-frame inspection.

[2] L/x_i is an integer since both L and x_i are powers of 2 multiples of the minimum non-zero element in $\mathcal{C}$ and $x_i \leq L$.

D. Theoretical Analysis

In this part, we analyze the approximation ratio on achieved system uncertainty by BPB, with the following theorem.

Theorem 1. *Let U be the overall system uncertainty under the BPB policy, $\tilde{U}^*$ be the optimal uncertainty caused by wait intervals, and U_f be the uncertainty caused by full-frame inspection latency, where $\tilde{U}^* \ll U_f$[3]. We use U^* to denote the optimal overall uncertainty. If the object weights $w_1, \ldots, w_N$ are integer powers of 2, then $U \leq (1 + \frac{2\tilde{U}^*}{U_f})U^*$; otherwise in general case, $U \leq (1 + \frac{4\tilde{U}^*}{U_f})U^*$.*

We reindex the objects in the decreasing order of their weight factors, *i.e.*, $w_1 \geq \cdots \geq w_N$. We first utilize the symmetric structure of the schedule computed by BAS (*i.e.*, the mapping of each subsequent task of an object is a duplicate of the first task to the corresponding subset of bins), to bound the uncertainty caused by wait intervals. Then, we include the uncertainty caused by inspection latency, and derive the bound for overall uncertainty. The proof consists of four steps:

- **Step 1:** The load difference $\bar{\lambda}_l(i) - \underline{\lambda}_l(i)$, between the max bin load $\bar{\lambda}_l(i) := \max_l \lambda_l(i)$ and the min bin load $\underline{\lambda}_l(i) := \min_l \lambda_l(i)$, is always bounded, where $\lambda_l(i)$ is the load for bin B_l after assigning the first i objects.
- **Step 2:** Given a inspection frequency set, BAS is optimal in minimizing the overall execution latency.
- **Step 3:** We prove the bound on the system uncertainty caused by wait intervals, by bounding the maximum bin load with the optimal system uncertainty.
- **Step 4:** We include the uncertainty caused by inspection latency, and prove the overall uncertainty bound.

Next, we go through the above steps one by one. Due to the space limitation, we skip the proof of some lemmas here.

Step 1: We first claim that the bin load difference is bounded at every step of BAS execution.

Lemma 1. *For each object $\mathcal{O}_i$, $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \max\{\bar{\lambda}(i-1) - \underline{\lambda}(i-1), \tau_{s_i}\}$, where s_i is the target size for the object $\mathcal{O}_i$ and τ_{s_i} is its corresponding batch execution time.*

Step 2: We give the optimality of BAS schedule in minimizing the system load of given inspection frequency set.

Lemma 2. *Given an inspection frequency set $\{x_1, x_2, \ldots, x_N\}$, we use λ_{BAS} to denote the total load of the schedule computed by the BAS algorithm (Algorithm 3). It minimizes the total load over all feasible schedules for the given inspection frequency set,* i.e., *$\lambda_{BAS} \leq \lambda$, with λ being the total load of any other feasible schedule.*

Step 3: We prove the bound on system uncertainty caused by wait intervals only.

Lemma 3. *Let $\tilde{U}^*$ be the optimal system uncertainty caused by wait intervals, and $\tilde{U}$ be that part in BPB policy, respectively.*

[3] We base on the assumption that it is beneficial to slice the image and run the inspection tasks at the sub-frame level.

If the object weights are all integer powers of 2, then $\tilde{U} \leq 2\tilde{U}^*$*; otherwise in general case,* $\tilde{U} \leq 4\tilde{U}^*$.

Proof. Let $\{\tilde{x}_1^*, \ldots, \tilde{x}_N^*\}$ be the inspection frequency set that achieves $\tilde{U}^*$. We construct its *proportional adaptation* using the following procedure.

- Let $T' = T - t_f$. Under the optimal schedule, we have $\tilde{U}^* \geq \max_i \frac{w_i T'}{\tilde{x}_i^*}$. Let $\hat{i} = \arg\max_i \frac{w_i T}{\tilde{x}_i^*}$. For each i, $\frac{w_i}{\tilde{x}_i^*} \leq \frac{w_{\hat{i}}}{\tilde{x}_{\hat{i}}^*}$. Since each $\tilde{x}_i^*$ is an integer, it follows that $\tilde{x}_i^* \geq \frac{w_i \tilde{x}_i^*}{w_{\hat{i}}} \geq \left\lfloor \frac{w_i \tilde{x}_i^*}{w_{\hat{i}}} \right\rfloor$. We set $\hat{c} = \left\lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \right\rfloor \geq 1$.[4]
- Let $x_i = 2^{\lfloor \log_2(w_i/w_N) \rfloor}$, that is, $\{x_1, \ldots, x_N\}$ is the output of step 3 of **Algorithm** 2, *i.e.*, the ratios of the inspection frequency set of BPB. By definition, $x_N = 1$. According to the construction, for each i, we have $x_i = 2^{\lfloor \log_2(w_i/w_N) \rfloor} \leq \left\lfloor \frac{w_i}{w_N} \right\rfloor$.
- We define $\{\hat{c}x_1, \ldots, \hat{c}x_N\}$ as the proportional adaptation of the optimal inspection frequency set. Since both $\hat{c}$ and x_i are both integers, we have $\lfloor \hat{c}x_i \rfloor = \hat{c}x_i$.

We next prove that the constructed proportional adaptation is feasible that can finish within T'. For each object $\mathcal{O}_i$,

$$\hat{c}x_i \leq \hat{c} \left\lfloor \frac{w_i}{w_N} \right\rfloor = \left\lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \right\rfloor \left\lfloor \frac{w_i}{w_N} \right\rfloor \leq \left\lfloor \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \right\rfloor \leq \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \leq \tilde{x}_i^*$$

Since the optimal schedule is feasible, there also exists a feasible schedule for the inspection frequency set $\{\hat{c}x_1, \ldots, \hat{c}x_N\}$.

We have proved (Lemma 2) that BAS minimizes the system load, thus the factor c by BAS is at least $\hat{c}$, *i.e.*, $c \geq \hat{c}$. In the BPB policy, the object uncertainty is bounded by $w_i(\max_l \lambda_l)\frac{L}{x_i} = w_i(\max_l \lambda_l)\frac{x_1}{x_i}$, where λ_l is the load of bin B_l. We bound the maximum bin load of the BPB schedule, under the following two cases.

(Case 1): If $\bar{\lambda}(N) \leq 2\underline{\lambda}(N)$, we have

$$\max_l \lambda_l = 2 \min_l \lambda_l \leq \frac{2\lambda_{BAS}}{L} \leq \frac{2T'}{cx_1} \leq \frac{2T'}{\hat{c}x_1},$$

From the construction of $\{x_1, \ldots, x_N\}$, we have for each object $\mathcal{O}_i$, $\frac{x_i}{x_N} \leq \frac{w_i}{w_N} \leq \frac{2x_i}{x_N}$. Its uncertainty satisfies,

$$\frac{w_i x_1}{x_i} \cdot \max_l \lambda_l \leq \frac{w_i x_1}{x_i} \cdot \frac{2T'}{\hat{c}x_1} \leq \frac{4w_N T'}{\hat{c}x_N} = \frac{4w_N T'}{w_N \tilde{x}_{\hat{i}}^* / w_{\hat{i}}} \leq 4\tilde{U}^*.$$

(Case 2): If $\bar{\lambda}(N) > 2\underline{\lambda}(N)$, consider the last i where $\bar{\lambda}(i)$ increases (*i.e.*, $\bar{\lambda}(i) > \bar{\lambda}(i-1)$, we have $\bar{\lambda}(N) - \underline{\lambda}(N) \leq \bar{\lambda}(i) - \underline{\lambda}(i) \leq \tau_{s_i}$. We have $\tau_{s_i} \geq \max_l \frac{\lambda_l}{2}$. Even under the optimal schedule, the maximum uncertainty of object $\mathcal{O}_1$ is at least $w_1 \tau_{s_i} \leq \tilde{U}^*$, so we have $\max_l \lambda_l \leq \frac{2\tilde{U}^*}{w_1}$. Hence,

$$\frac{w_i x_1}{x_i} \cdot \max_l \lambda_l \leq \frac{w_i x_1}{x_i} \cdot \frac{2\tilde{U}^*}{w_1} \leq \frac{2w_N}{x_N} \cdot \frac{x_N}{w_N} \cdot 2\tilde{U}^* = 4\tilde{U}^*.$$

Specially, if each w_n is integer power of 2, we have $\frac{x_i}{x_N} = \frac{w_i}{w_N}$, then it holds $\tilde{U} \leq \tilde{U}^*$ in both cases. □

[4]Without loss of generality, we assume that $\left\lfloor \frac{w_N \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \right\rfloor \geq 1$ and $\frac{\tilde{x}_{\hat{i}}^*}{w_{\hat{i}}}$ is an integer; otherwise, we can just take the largest $\hat{i}$ with non-zero value of this equation and leave out the remaining objects.

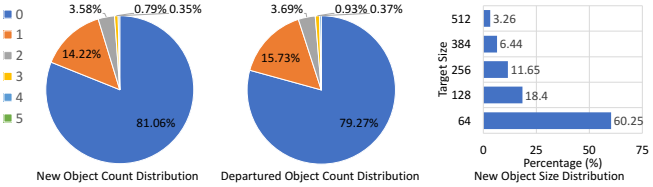


Fig. 6. Distributions on number of **newly arrived objects** and **departured objects**, as well as the **(quantized) new object size distribution**, at each frame. Results obtained on Waymo Open dataset [43].

Fig. 7. An example of new pixels in the current frame highlighted in red.

Step 4: We prove the bound on the overall system uncertainty, including uncertainty caused by inspection latencies.

Proof. We use t_f to denote the full-frame inspection latency. Recall that $\tilde{U}^*$ is the optimal uncertainty caused by inspection intervals, U^* is the optimal overall uncertainty, and $\tilde{U}^* \leq U^*$. They can correspond to two different schedules. We have,

$$\begin{aligned} U = \max\{U_1, \ldots, U_N\} &\leq \max\{\tilde{U}_1 + w_1 t_f, \ldots, \tilde{U}_N + w_N t_f\} \\ &\leq \max\{\tilde{U}_1, \ldots, \tilde{U}_N\} + U_f \\ &\leq 4\tilde{U}^* + U_f = (\frac{4\tilde{U}^*}{U_f} + 1)U_f \\ &\leq (\frac{4\tilde{U}^*}{U_f} + 1)U^*. \end{aligned}$$

Since every schedule includes the full-frame inspection, which induces uncertainty U_f, we have $U_f \leq U^*$. The proof follows. Similarly, when all w_n's are integer power of 2, we have $U \leq (1 + \frac{2\tilde{U}^*}{U_f})U^*$. This completes the proof of Theorem 1. □

VI. Empirical Optimization

In this section, we list some practical considerations and empirical optimizations we performed in our implementation.

1) New Object Arrival: We first show in Figure 6 that there is no object arrival or departure in most ($\approx 80\%$) frames. Most new objects have very small sizes so they only cause minor extra workload. Some existing objects can disappear during the scheduling horizon. The slots for these objects, together with the idle slots in incomplete batches, can be used to schedule the new object regions. To (roughly) localize new objects, we apply a lightweight mechanism based on optical flow. We define the pixels in the new frame that are not mapped to any pixel in the previous frame as the newly appeared pixel, and then use connected component analysis [44] to extract new object regions. An example is shown in Figure 7.

2) Downsizing Large Objects: For the tracked large objects, we can safely downsize their resolutions without affecting the detection quality, because large objects are known to be easy to detect [45], [46]. We set an upper bound (*e.g.*, 256) on the

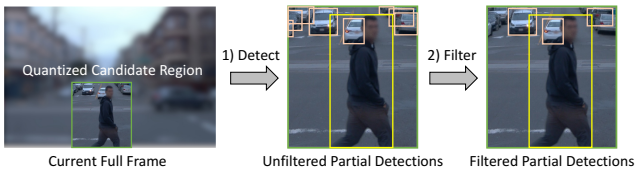


Fig. 8. An example of bounding box filtering. Note we preserve the yellow box although its bottom edge lies on the bottom border of the partial image, because the partial image bottom coincides with the full image bottom.

target sizes, where the candidate regions larger than this size will be downsized and fit into the size.

3) Partial Region Merge: If two candidate regions are highly overlapped, it is beneficial to merge them into one, so we can avoid repetitively scanning the same area. In our case, if there is an unscheduled region that its overlap ratio with a scheduled region is above a threshold I, we use the merged region to replace the scheduled region.

4) Bounding Box Filtering: We perform a bounding box filtering procedure, as a postprocessing step, to remove fragmented detections that correspond to only part of a physical object. Specifically, we will remove detected bounding boxes that lie on the partial image boundaries, unless the partial image boundaries coincide with the full image boundaries. We provide an illustrative example in Figure 8. Intact redundant inspections can be easily removed by the non-maximum suppression (NMS) step of the detector.

VII. Evaluation

In this section, we evaluate the effectiveness and efficiency of the proposed architecture on an NVIDIA Jetson Xavier board with a real-world self-driving dataset.

A. Experimental Setup

1) Hardware Platform: All experiments are conducted on an NVIDIA Jetson Xavier SoC, which is designed for automotive platforms. It is equipped with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory. The mode is set as MAXN.

2) Dataset: Our experiment is performed on the Waymo Open Dataset [43], a large-scale autonomous driving dataset collected by Waymo self-driving cars in diverse geographies and conditions. It consists of driving video segments of 20s each, collected by onboard cameras at 10Hz with resolution 1920×1280. Only front camera data is used.

3) Neural Network for Detection: We use the YOLOv5[5] model in PyTorch as the object detection network, which was pretrained on the general-purpose COCO [47] dataset. We specifically use the default "large" config in the evaluation, with both depth and width multipliers set to 1. The model precision is set to FP16. The YOLO inference latency with different target sizes are profiled in advance.

4) Workload Manipulation: Unless otherwise indicated, we choose our scheduling horizon to be 10 frames, and manually change the time interval P between two consecutive frame arrivals to induce different workload. Intuitively, a shorter frame interval leads to a higher scheduling load. Our experiments use three interval lengths (150ms, 100ms, and 70ms) to denote the easy, moderate, and hard scheduling situations (corresponding to frame rates of roughly 6.67Hz, 10Hz, and 14Hz).

5) Object Criticality: The object criticality is the product of two terms: 1) Class criticality, 2) Approximated object distance. The *class criticality* is manually assigned to simulate how humans prioritize different types of object. For example, "human" class has a much higher criticality than "vehicle" class. Besides, we assume the physical sizes of objects belonging to one class are similar, so we use the bounding box size (*i.e.*, width) as an approximation of object distance. We separately evaluate the detection performance on *all objects* and *critical objects*. A separate object size threshold for critical objects is set for each class.

6) Evaluation Metrics: Our metrics distinguish between performance of *detection*, *localization*, and *classification*. Here *detection* means discovering whether an object exists (at a location) or not, regardless of type. *Localization* means identifying the position of the object. Finally, *classification* is the process of identifying object type. Given a list of detections and a list of groundtruth object locations, we match the detections with the groundtruth objects based on their bounding box overlaps. A detection is said to be matched with a groundtruth object if their IoU ratio is larger than a predefined threshold (set as 0.5 in this paper), in which case we say that the object is *successfully detected*. The following set of metrics are then defined:

- **Detection Recall (DR):** The ratio between the number of successful detections (matched with groundtruth objects) and the count of all groundtruth objects.
- **Detection Precision (DP):** The ratio between the number of successful detections (matched with groundtruth objects) and the count of all detections.
- **Classification Accuracy (CA):** For each successful detection, we test whether the predicted object class is correct and report ratio of correct classifications.
- **Localization Error (LE):** For each successful detection, its location error is the distance between the estimated and ground truth object center points, as fraction of the object size (*i.e.*, diagonal length).
- **Mean Average Precision (mAP):** It is used as an end-to-end metric, which simultaneously captures the error in both location and classification. An open sourced mAP evaluation engine[6] is used.

The YOLO performance on full frames is listed in Table I, which serves the ceiling condition for the proposed framework.

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

[6]https://github.com/Cartucho/mAP

TABLE I
YOLOV5 PERFORMANCE ON WAYMO DATASET. ALL VALUES IN THIS TABLE ARE IN PERCENT, EXCEPT THE LATENCY.

Model	Ove. Det. Rec.	Ove. Det. Pre.	Ove. Cls. Acc.	Ove. Loc. Err.
YOLOv5l	70.09	87.54	99.88	4.68
	Cri. Det. Rec.	Cri. Det. Pre.	Cri. Cls. Acc.	Cri. Loc. Err
	82.29	92.05	99.96	3.97
	Ove. mAP	Cri. mAP	Xavier Latency	
	62.76	78.14	239ms	

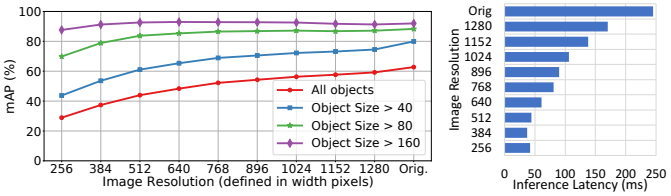


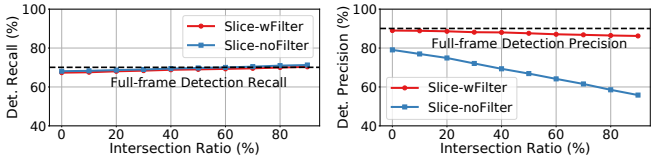
Fig. 9. Impact of image resolution on detection quality and inference latency.

B. Impact of Image Downsizing

One intuitive question is, *why not just downsize the images such that full-frame inspections can run in real time*. In this section, we investigate the impact of image downsizing on achieved detection quality and inference latency. We downsize the images to different resolutions, and evaluate the mAP on objects of different sizes, as well as the corresponding latency. The results are shown in Figure 9. We have the following observations: First, reducing image resolutions results in degraded detection quality, especially on small objects. However, recognizing small objects is still important in autonomous driving. Second, large objects are more robust to image downsizing. After image slicing, we can safely reduce the resolution of candidate regions for large objects to achieve better efficiency. Therefore, we set 256 as the largest target size, where larger objects are downsized and fit into it.

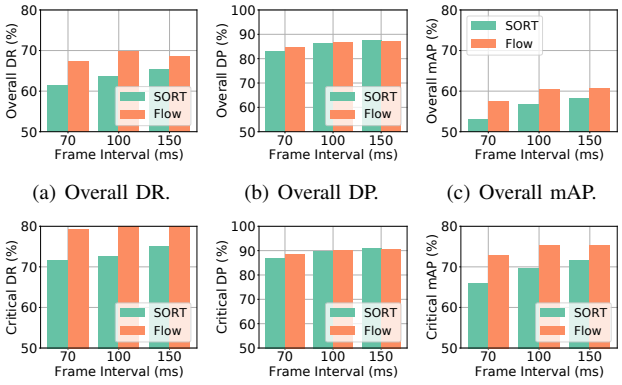
C. Impact of Image Slicing

A good slicing module should be lossless and lead to no degradation in detection quality. To isolate the impact of image slicing, we run inspections on all sliced candidate regions. Besides, two empirical optimizations are considered: (i) bounding box filtering, and (ii) candidate region merge. We evaluate the detection recall and precision with/out bounding box filtering, under different candidate region merge criteria (*i.e.*, the intersection ratios) in Figure 10. First, the detection precision is degraded after slicing, because more false positive detections (*i.e.*, fragmented object parts) are generated. The region merge does help partially improve detection precision, but bounding box filtering is the key factor that makes the slicing lossless. The red curve of Figure 10(b) indicates the slicing with bounding box filtering shows negligible degradation on detection precision under different merging criteria. The fragmented detections are mostly removed. Second, the detection recall is not affected no matter whether bounding box filtering is applied, which indicates the sliced partial frames completely cover the groundtruth objects. We set the intersection ratio for merge as 0.5, to achieve a good tradeoff between detection recall and precision.



(a) Detection recall. (b) Detection precision.

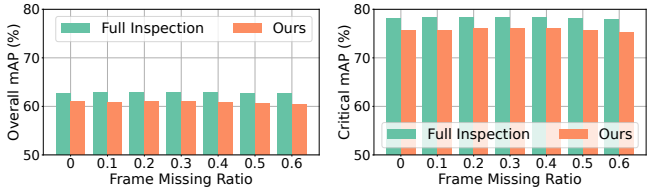
Fig. 10. Impact of slicing on detection quality.



(a) Overall DR. (b) Overall DP. (c) Overall mAP.

(d) Critical DR. (e) Critical DP. (f) Critical mAP.

Fig. 11. The impact of tracking algorithms on the detection quality of overall objects and critical objects.



(a) Overall mAP. (b) Critical mAP.

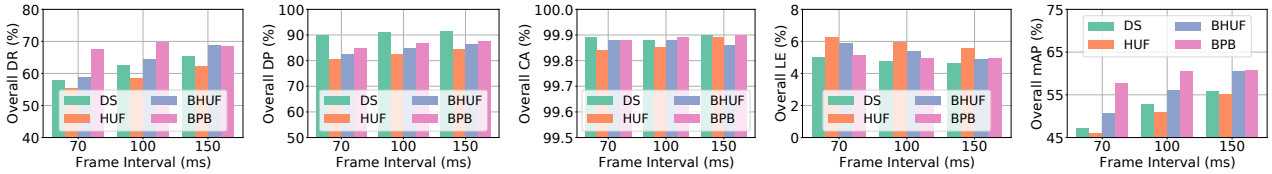
Fig. 12. The impact of missing frames on our slicing and partial inspection approach.

D. Tracking Algorithm

We compare our flow-based tracker (denoted by "`Flow`") with a state-of-the-art tracking algorithm, `SORT` [40], which uses a Kalman filter to model object motions. It extrapolates future object locations from the past object trajectories. The results are presented in Figure 11. We separately show the results on *overall objects* and *critical objects*, under three workloads (*i.e.*, frame intervals). We found that optical flow generally works better than SORT in tracking. They show similar detection precision under each workload, but the detection recall and mAP of `Flow` are clearly better than `SORT`, especially when the frame interval is short. We rely more on the tracking algorithm to predict object locations when there is no GPU resource to run their partial-frame detection tasks. `Flow` is more accurate in estimating object motions, because it proactively extracts the information from newly captured frames, as opposed to the extrapolated motions in `SORT`.

E. Robustness of Flow-based Tracking and Slicing

In this experiment, we explicitly evaluate the robustness of the proposed flow-based tracking algorithm, by answer-



(a) Overall DR. (b) Overall DP. (c) Overall CA. (d) Overall LE. (e) Overall mAP.

(f) Critical DR. (g) Critical DP. (h) Critical CA. (i) Critical LE. (j) Critical mAP.

Fig. 13. Scheduling algorithms comparison. The first row shows detection results on overall objects, and the second row shows results on critical objects.

ing the following question: Does missing frames during a scheduling horizon affect the detection result? We randomly delete different portions (from 10% to 60%) of frames from each scheduling horizon (*i.e.*, 10 frames), and compare the relative detection performance between our approach and full-frame inspection on the remaining frames. All sliced candidate regions are inspected. If the key frame is missing, we regard the next available frame as the key frame instead. The results are summarized in Figure 12. We use reproducible random number generator to generate the same subset of missing frames between the two compared approaches. Our flow-based tracking and slicing approach consistently shows a close performance on both overall mAP and critical mAP to the full-frame inspection approach, when different portions of frames are missing within a scheduling horizon. Only negligible relative degradation is observed as the frame missing ratio increases. When intermediate frames are missing, the optical flow algorithm would directly compute the flow map between two consecutive available frames, and the proposed expansion steps further consider the potential uncertainty contained in the estimated flow map.

F. Scheduling Algorithm Comparison

1) Baselines: We compare with the following algorithms.

- **Downsizing (DS):** It always runs full-frame inspections at the largest resolution that can finish in real-time.
- **Highest Uncertainty First (HUF):** It always schedules the partial frame inspection task with the highest weighted uncertainty. Batching is not used.
- **Batched Highest Uncertainty First (BHUF) [1]:** It always schedules the partial frame inspection tasks with the highest weighted uncertainty, and batches the tasks under the same target size in a greedy manner.

2) Results: The corresponding results are summarized in Figure 13. We test the scheduling algorithms at different workloads (*i.e.*, frame intervals). We report the following observations: First, note that the new algorithm (BPB) substantially improves mAP, a metric that captures both location and classification errors. This is consistent with the goal of this work to improve location accuracy (without hurting classification accuracy). Second, BPB does not do worse on other metrics. For example, although DS maintains a similar localization error, it does worse on recall because it loses small objects. BPB achieves the best recall, but at the cost of minor degradation in precision. One can argue that in autonomous driving, recall is more important than precision, because false negatives are more of a safety problem than false positives. Furthermore, compared to BHUF, BPB does better at planning when to invoke the detector to minimize location error without hurting recall. As a result, BPB achieves both a lower location error and a higher recall, especially at higher frame rates (*i.e.*, when $P = 70ms$). Finally, there are almost no misclassifications of object types with BPB, and the localization error roughly ties on lowest.

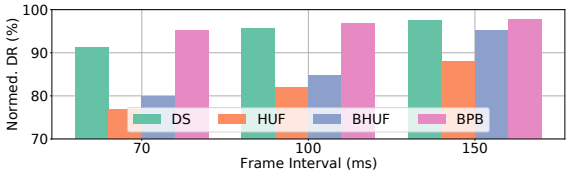


Fig. 14. The normalized detection recall on physically close objects.

G. Responsiveness to Physically Close Objects

We evaluate the achieved detection recall on all objects within 30 meters to the ego-vehicle (according to data set ground truth). The results are normalized by the detection recall achieved on full frames, and reported in Figure 14. The proposed BPB algorithm outperforms the baselines, especially when the frame interval is short. The evaluation demonstrates that the absence of a physical ranging sensor is not a hindrance and that (visual) size-based assignment of priority offers higher recall on close objects compared to baselines such as whole image resizing.

H. Breakdown of Overhead Quantification

Next, we report the breakdown latency overhead induced by our framework. The results are shown in Figure 15. Since

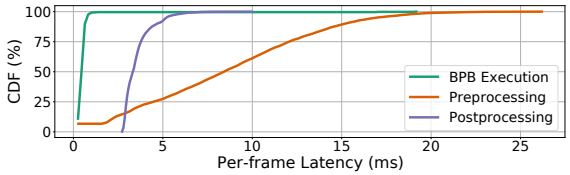
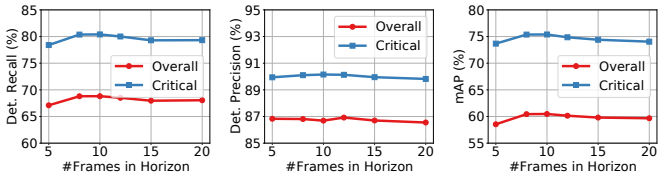


Fig. 15. Breakdown latency overhead in the proposed framework.



(a) Detection Recall. (b) Detection Precision. (c) mAP.

Fig. 16. The impact of the scheduling horizon length on detection quality.

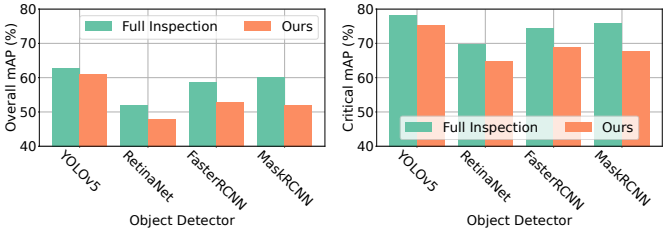
the BPB policy only executes once per scheduling horizon, its overhead is divided into each frame, which turns out to be minimized in our implementation. The preprocessing steps include new object localization, image slicing, candidate region merge and batching at each frame, while the postprocessing steps filter the generated detections and map the remaining detections to the full frame coordinates. The optical flow estimator runs in an independent process on the CPU and poses no overhead to the detection pipeline on GPU. Both preprocessing and postprocessing overhead is very small in most cases. We also notice the preprocessing delays are relatively high in some heavy traffic scenarios, where a large amount of candidate regions are extracted.

I. Choice of Scheduling Horizon Length

Here, we investigate the impact of the scheduling horizon length in BPB policy. We change the horizon length from 5 to 20, and see how the detection quality are affected. We set the frame arrival interval $P = 100ms$. The results are summarized in Figure 16. Our BPB policy is generally resilient to the horizon length, and does not show a large variation on achieved detection quality. The moderate lengths (8 or 10) show sightly better detection recall on both types of objects. When the scheduling horizon is too short, we only have very limited time to schedule partial-frame inspection tasks. Too much time is spent on full-frame inspections. When the scheduling horizon is too long, we do not have timely updates on the object presence and criticality, and may waste time tracking objects that are not critical anymore. Thus, shorter horizons ensure better freshness on object list, while longer horizons provide more scheduling flexibility. We believe choosing a moderate length (*i.e.*, 10 frames or 1 second) is most beneficial that achieves a good tradeoff between scheduling flexibility and freshness on object list.

J. Impact of Object Detector

Finally, we evaluate the impact of the object detection model. In addition to YOLOv5, we use three representative object detectors: RetinaNet [48], FasterRCNN [49], and MaskRCNN [50]. To evaluate the model impact, we use 60% of the full-frame inference latency of each model as the simulated frame sampling period. The achieved overall and critical mAPs are presented in Figure 17. We found our approach generally works with different models, which results in 2.1% to 8.1% degradation on overall mAP, and 2.8% to 8.3% degradation on critical mAP. The degradation differs among models because of their difference in computation parallelizability. To illustrate this fact, we show how the batch latency increases with the batch size in Figure 18 with 256×256 input. YOLOv5 is the state-of-the-art model with highly optimized GPU implementation, so we can observe a flat curve when the batch size is within 5. Comparatively, RetinaNet, although a single-stage detector, has serialized implementation in its preprocessing steps. The parallelizability of FasterRCNN and MaskRCNN are even worse because of the two-stage nature in their design, which divide the computation into the region proposal and classification. More serialized execution is included in the classification step, thus the associated overhead may partially consume the efficiency saving by slicing and batching.



(a) Overall mAP. (b) Critical mAP.

Fig. 17. The impact of object detection models on our approach.

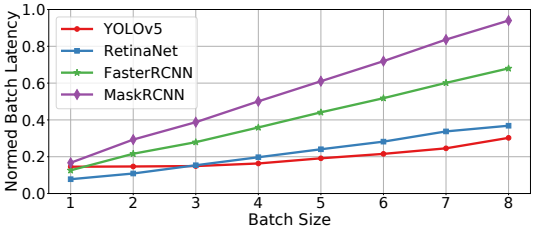


Fig. 18. Batch latency curve for different object detectors with input size 256×256. The batch latency is normalized by the model full-frame latency.

VIII. Conclusions

We described a self-cueing attention scheduling framework to optimize the efficiency of visual machine perception (on resource-limited embedded platforms) at minimizing location error without hurting recall. A scheduling algorithm with a theoretically proven approximation ratio (in terms of maximum location uncertainty) was described and implemented on an NVIDIA Jetson Xavier board. Empirical evaluation using a real-world driving dataset indicates the feasibility of the self-cueing approach. The work advances attention scheduling literature by allowing AI-based perception pipelines to selectively schedule data processing *at the subframe level* (consistently with tracking and/or safety needs) without external cueing.

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. 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, or the US government. The US government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *In Proc. IEEE Real-time Systems Symposium (RTSS)*, December 2020.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," pp. 779–788, 2016.

[3] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, "Self-driving cars," *Computer*, vol. 50, no. 12, pp. 18–23, 2017.

[4] J. Scott and C. Scott, "Drone delivery models for healthcare," in *Proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[5] W.-Y. G. Louie, T. Vaquero, G. Nejat, and J. C. Beck, "An autonomous assistive robot for planning, scheduling and facilitating multi-user activities," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5292–5298.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[10] Z. Huang, Z. Chen, Q. Li, H. Zhang, and N. Wang, "1st place solutions of waymo open dataset challenge 2020–2d object detection track," *arXiv preprint arXiv:2008.01365*, 2020.

[11] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 278–291.

[12] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] J. Huang, C. Samplawski, D. Ganesan, B. Marlin, and H. Kwon, "Clio: enabling automatic compilation of deep learning pipelines across iot and cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–12.

[14] 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.

[15] S. Lee and S. Nirjon, "Fast and scalable in-memory deep multitask learning via neural weight virtualization," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 175–190.

[16] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017, p. 4.

[17] B. Minnehan and A. Savakis, "Cascaded projection: End-to-end network compression and acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 715–10 724.

[18] Z. Yang, K. Nahrstedt, H. Guo, and Q. Zhou, "Deeprt: A soft real time scheduler for computer vision applications on the edge," *arXiv preprint arXiv:2105.01803*, 2021.

[19] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "Gpu scheduling on the nvidia tx2: Hidden details revealed," in *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2017, pp. 104–115.

[20] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 119–130.

[21] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[22] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *In Proc. IEEE Real-time Systems Symposium (RTSS)*, December 2020.

[23] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[24] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time dnn inference and training," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 15–29.

[25] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 174–187.

[26] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *In Proc. IEEE International Conference on Embedded and Real-time Computing Systems and Applications (RTCSA)*, August 2020.

[27] 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.

[28] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2021, pp. 169–178.

[29] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep feature flow for video recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2349–2358.

[30] S. Wang, H. Lu, and Z. Deng, "Fast object detection in compressed video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7104–7113.

[31] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-guided feature aggregation for video object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 408–417.

[32] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: Principled cache for mobile deep vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 129–144.

[33] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "Eva^2: Exploiting temporal redundancy in live computer vision," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 533–546.

[34] L. Cavigelli, P. Degen, and L. Benini, "Cbinfer: Change-based inference for convolutional neural networks on video data," in *Proceedings of the 11th International Conference on Distributed Smart Cameras*, 2017, pp. 1–8.

[35] S. Zhang, W. Lin, P. Lu, W. Li, and S. Deng, "Kill two birds with one stone: Boosting both object detection accuracy and speed with adaptive patch-of-interest composition," in *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2017, pp. 447–452.

[36] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 1010–1021.

[37] A. R. Kumar, B. Ravindran, and A. Raghunathan, "Pack and detect: Fast object detection in videos using region-of-interest packing," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2019, pp. 150–156.

[38] H. Mao, T. Kong, and W. J. Dally, "Catdet: Cascaded tracked detector for efficient object detection from video," *arXiv preprint arXiv:1810.00434*, 2018.
[39] 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.
[40] 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.
[41] A. Torralba, "How many pixels make an image?" *Visual neuroscience*, vol. 26, no. 1, pp. 123–131, 2009.
[42] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The pinwheel: A real-time scheduling problem," in *Proceedings of the 22nd Hawaii International Conference of System Science*, 1989, pp. 693–702.
[43] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.
[44] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1596–1609, 2010.
[45] T.-W. Chin, R. Ding, and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling," in *Systems and Machine Learning Conference*, 2019.
[46] R. Xu, C.-l. Zhang, P. Wang, J. Lee, S. Mitra, S. Chaterji, Y. Li, and S. Bagchi, "Approxdet: content and contention-aware approximate object detection for mobiles," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 449–462.
[47] 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.
[48] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.
[49] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," pp. 91–99, 2015.
[50] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.