Real-Time Systems (2023) 59:302–343
https://doi.org/10.1007/s11241-023-09396-z

Generalized self-cueing real-time attention scheduling with intermittent inspection and image resizing

Shengzhong Liu[1] · Xinzhe Fu[2] · Yigong Hu[1] · Maggie Wigness[3] · Philip David[3] · Shuochao Yao[4] · Lui Sha[1] · Tarek Abdelzaher[1]

Accepted: 25 March 2023 / Published online: 8 June 2023

Abstract

This paper proposes a generalized self-cueing real-time attention scheduling framework for DNN-based visual machine perception pipelines on resource-limited embedded platforms. Self-cueing means we identify subframe-level regions of interest in a scene *internally* by exploiting temporal correlations among successive video frames as opposed to externally via a cueing sensor. One limitation of our original self-cueing-and-inspection strategy (Liu et al. in Proceedings of the 28th IEEE real-time and embedded technology and applications symposium (RTAS), 2022b) lies in its lack of computational efficiency under high workloads, like busy traffic scenarios where a large number of objects are identified and separately inspected. We extend the conference publication by integrating *image resizing* with *intermittent inspection* and *task batching* in attention scheduling. The extension enhances the original algorithm by accelerating the processing of large objects by reducing their resolution at the cost of only a negligible degradation in accuracy, thereby achieving a higher overall object inspection throughput. After extracting partial regions around objects of interest, using an optical flow-based tracking algorithm, we allocate computation resources (i.e. DNN inspection) to them in a criticality-aware manner using a generalized batched proportional balancing algorithm (GBPB), to minimize a concept of *generalized system uncertainty*. It saves computational resources by inspecting low-priority regions intermittently at low frequencies and inspecting large objects at low resolutions. We implement the system on an NVIDIA Jetson Xavier platform and extensively evaluate its performance using a real-world driving dataset from Waymo. The proposed GBPB algorithm consistently outperforms the previous BPB algorithm that only uses intermittent inspection and a set of baselines. The performance gain of GBPB is larger in facing more significant resource constraints (i.e., lower sampling intervals or busy traffic scenarios) because its multi-dimensional scheduling strategy achieves better resource allocation of machine perception.

Keywords Real-time scheduling · Object detection · Temporal correlations · Cyber-physical systems

Extended author information available on the last page of the article

1 Introduction

Attention prioritization and scheduling of intelligent perception pipelines is a novel problem in real-time systems literature (Liu et al. 2020a) 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 data 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 extends a previous conference publication (Liu et al. 2022b) by proposing a *generalized self-cueing attention scheduling framework* for visual machine perception on embedded platforms. It generalizes the original self-cueing framework by simultaneously optimizing image resizing decisions along with the original intermittent inspection and task batching strategies. Compared to external-cueing frameworks (Liu et al. 2020b; Hu et al. 2021; Kang et al. 2022b) that rely on light-weight processing of an external cueing sensor (e.g. LiDAR depth clustering) to extract object regions of interest on image frames, self-cueing frameworks fundamentally remove the dependency on external cueing by exploiting temporal correlations between object locations in successive frames, thus cueing attention to expected future object locations given their recent past. The main workload to optimize is object detection [e.g. YOLO (Redmon et al. 2016)] and tracking that exploits deep neural networks (DNNs). Specifically, low-frequency full-frame inspections are performed periodically, at a period we call the *scheduling horizon* (i.e. once per second), to identify the locations of all objects. In between full-frame inspections, a set of partial-frame inspections are scheduled that are focused only on predicted locations of individual objects, thereby reducing computational demand (compared to full-frame inspections) and latency. We next explain how the two contributing dimensions in this paper, *intermittent inspection* and *image resizing*, open up a novel optimization space for self-cueing scheduling.

First, intermittent inspection selectively decides which objects to inspect in a frame instead of inspecting every object in every frame. As explained in Liu et al. (2022b), the period at which an object is inspected is determined by the uncertainty growth in its location in between inspections. To bound the location uncertainty of the object, faster or more erratically-moving objects are scheduled for inspection at higher frequencies, while relatively static or slow-moving objects are scheduled at lower frequencies.

Second, compared to the original framework, we further integrate image resizing to save the execution latency on each scheduled inspection task. One existing limitation of the original self-cueing-and-inspection strategy lies in its lack of computational efficiency in dealing with busy traffic scenarios when a large number of (potentially overlapped) regions are extracted and separately inspected. In

such scenarios, the processing demand can be even higher than inspecting full frames. With image resizing, we can reduce the spatial resolution of the extracted regions before feeding them into the DNN model. Smaller images are known to have a lower execution latency by DNN models, at the cost of possibly degraded detection accuracy. Image resizing is computationally lightweight, easy to implement, and does not require modifications or switching on the deployed DNN model. Appropriately downsizing the partial frames facilitates higher object inspection frequencies, but does not sacrifice much in detection accuracy. Comparatively, large objects are more tolerant of image resizing than small objects with more latency savings and less accuracy degradation. Accordingly, we differentiate the degrees of resizing for objects of different sizes.

To jointly model the impact of both optimization dimensions, we formulate the scheduling problem as one of minimizing the *maximum weighted generalized uncertainty*. The new optimization objective multiplies the object location uncertainty caused by intermittent inspection with the accuracy degradation factor caused by image resizing, along with an application-specific object weight indicating the object criticality. The underlying assumption behind this metric is that the errors resulting from the two sources of uncertainty are multiplicative. Consequently, there is a tradeoff between inspecting the objects at higher frequencies or at higher resolutions. The optimal choice depends on the specific perception scenario. For example, slow-moving small objects might benefit more from higher resolutions, as opposed to higher frequencies, whereas fast-moving large objects might benefit more from higher frequencies over higher resolutions. 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.

We correspondingly propose a generalized batched proportional balancing (GBPB) algorithm to approximately solve the formulated scheduling problem. At each scheduling horizon after the full-frame inspection, the GBPB algorithm jointly produces three parts of output: the target size for each object after resizing, the inspection frequency of each object, and the task batching decisions among objects. Task batching means inspecting multiple images of the same size in parallel on the GPU, which achieves higher execution efficiency than serialized execution of inspection tasks. To reduce the complexity of the decision space, which originally grows exponentially with the number of objects, we uniformly set an upper bound on target sizes (such that any object regions beyond a threshold will be downsized to this size) and set the object inspection frequencies proportional to their uncertainty growth rates. In addition, object inspection tasks are scheduled in a batch-aware load-balanced manner such that the objects are inspected as many times as possible. At runtime, the algorithm adaptively trades off between object inspection frequencies and inspection qualities (i.e. target sizes) such that the resulting generalized maximum uncertainty can be minimized while the system resources are maximally utilized. Compared to the BPB policy in Liu et al. (2022b), GBPB achieves higher accuracy for high workloads under significant resource constraints (i.e. short frame intervals or busy traffic scenarios) but sustains a similar performance in light workload, because we avoid extremely low inspection frequencies on critical objects by compressing computation and saving time with effective image resizing. The DNN

processing resources are better allocated and maximally utilized when using the new multi-dimensional scheduling strategy.

We implement the proposed framework on an NVIDIA Jetson Xavier platform and extensively evaluate its performance using real-world driving datasets from Waymo (Sun et al. 2020). The results show that the proposed policy achieves a higher detection, localization, and classification quality (compared to baselines) under different workloads. It also provides better response times to physically close objects. In addition, the GBPB policy with image resizing integrated outperforms the original batched proportional balancing (BPB) policy under the same time constraints in busy traffic scenarios.

The rest of this paper is organized as follows: In Sect. 2, we briefly review the related literature. We give an overview of the architecture in Sect. 3, then explain the data slicing module in Sect. 4. We introduce the original BPB algorithm in Sect. 5, and further integrate the image resizing in Sect. 6. We discuss some adopted empirical optimizations in Sect. 7, before presenting the evaluation results in Sect. 8. Finally, we conclude the paper in Sect. 9.

2 Related work

2.1 Real-time machine perception

Most previous research to support real-time machine perception focused on compressing neural networks to reduce the inference latency (Yao et al. 2018; Zhou et al. 2018; Lee and Nirjon 2020a; Yao et al. 2020b; Minnehan and Savakis 2019). 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 (Yang et al. 2021). 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 (Amert et al. 2017; Capodieci et al. 2018; Xiang and Kim 2019; Jang et al. 2020; Kang et al. 2021; Amert et al. 2021) or utilize specialized DNN accelerators Restuccia and Biondi (2021). Besides, there have been works on optimizing the predictability (Liu et al. 2022a) and resource efficiency (Ji et al. 2022; Razavi et al. 2022) of DNN executions. In contrast, model-level scheduling algorithms dynamically adjust the utilized neural network structures to meet inference deadlines (Bateni and Liu 2018; Lee and Nirjon 2020b; Heo et al. 2020; Yao et al. 2020a; Kannan and Hoffmann 2021). Finally, the data-level scheduling algorithms change the amount of data to be processed by data scaling (Heo et al. 2022) or slicing the data into partial regions and processing them in a fine-grained and criticality-aware manner (Soyyigit et al. 2022). One drawback of existing approaches (Liu et al. 2020a, 2021; Hu et al. 2021, 2022; Kang et al. 2022b) lies in their reliance on an external attention cueing sensor (e.g. a ranging LiDAR), which might not be an option in some autonomous systems. For example, some autonomous car manufacturers, such as Tesla, famously objected to having LiDAR. In this paper, we

therefore build a self-cueing system that relies only on the original data flow without needing external secondary sensors. Different from Kang et al. (2022a) that targets scheduling multiple tracking tasks, we focus on fine-grained object tracking scheduling within a single task.

2.2 Temporal correlations in video object detection

Our self-cueing scheme fundamentally relies on object permanence to hypothesize that objects 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 (Zhu et al. 2017b; Wang et al. 2019; Zhu et al. 2017a; Xu et al. 2018), 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. (2018) proposed an optical flow-based hardware solution to propagate latent features from previous frames to the new frame. The uncertainty in the estimated motion is not counted. Some work also leverages pairwise image differences to guide an object detector to focus on changing areas in the new frame (Cavigelli et al. 2017; Zhang et al. 2017). However, they are only applicable to statically mounted cameras, which no longer work in autonomous driving systems. Song et al. (2020) applied different quantization levels to process regions with different sensitivity on the same frame, which was limited to image classification models only. Both Kumar et al. (2019) and Mao et al. (2018) 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 input into our attention prioritization and scheduling problem. The idea is generally applicable and can also be extended into mobile offloading scenarios (Liu et al. 2022d) and multi-camera collaborative sensing when spatial correlations are also considered (Liu et al. 2022c).

2.3 Dynamic DNN acceleration with image resizing

Image resizing has been used as one of the major controllable dimensions for dynamic latency saving in DNN execution on resource-limited platforms. Downsizing images accelerates inspection at the cost of different degrees of accuracy degradation. Downsizing is easy to implement, and DNN object detection models typically present sublinear accuracy degradation with respect to the downsizing ratio. Multiple efforts have proposed to adaptively downsize input images to catch up with runtime execution deadlines (Hu et al. 2021; Chin et al. 2019; Wu et al. 2022; Bastani and Madden 2021; Heo et al. 2022), or detect objects of different sizes at appropriate scales that optimize the model accuracy/latency trade-off (Najibi et al. 2019; Li et al. 2021). AdaScale (Chin et al. 2019) also shows that appropriately downsizing the images may even slightly improve the general model accuracy in detecting large objects in the scene. Both Hu et al.

(2021) and Heo et al. (2022) adopt image resizing as the control knob to achieve different tradeoffs between model accuracy and inference latency in the context of real-time edge AI. To the best of our knowledge, we are the first that formally study the cooperation of intermittent inspection and image resizing in achieving real-time machine perception on embedded platforms, which jointly achieve better accuracy-latency tradeoff than either knob alone.

3 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 (e.g., 1920×1280 resolution) can not finish before the next frame arrives. Instead, we inspect full frames at a longer interval T (say, 1-2s). We refer to processing of full frames as *full-frame inspections*. Between them, we identify regions of interest using optical flow (Kroeger et al. 2016), 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 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 object detector and how large these regions should be set for exact localization of the corresponding objects. We call the approach that integrates intermittent inspection and image resizing as *generalized 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 *generalized partial-frame inspection scheduling module*. An overview of the architecture is shown in Fig. 1.

3.1 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.

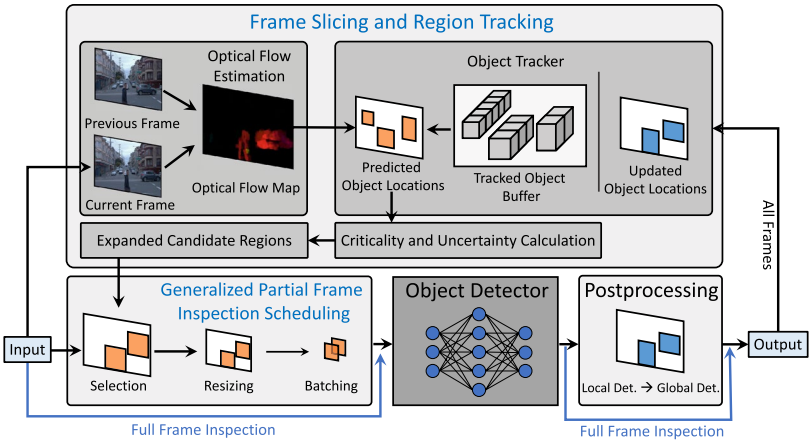


Fig. 1 The overview of the proposed scheduling framework. It is generally applicable to any object detector

3.2 Generalized partial-frame inspection scheduling

This model jointly decides how often each object should be inspected and how large its partial frame should be set after resizing, in order to achieve optimized tracking accuracy on each object. Every object is associated with a criticality indicating its application-level importance, and an uncertainty growth rate indicating the increase speed of its location uncertainty if no new inspection is performed. Partial-frame inspections of objects with low criticality or slow uncertainty growth are scheduled less frequently. In addition, we can safely reduce the image sizes for large objects without losing too much on their detection accuracy. Finally, we consider *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 achieves much lower latency than serialized processing.

4 Frame slicing and region tracking

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

4.1 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 (Kroeger et al. 2016), 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 reduce false positives (similar to the way Kalman filters leverage both imperfect models and imperfect empirical observations to produce improved trajectory estimates). We leave the integration of physical models as a future direction.

4.2 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 inferred object location for further DNN inspection without needing multiple prior observations (e.g., to form a trajectory). 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 (Bewley et al. 2016)], because the latter models often require a sequence of past observations to correctly estimate trajectories, and may fail to correctly predict location in the presence of sudden unexpected movements (such as swerves to avoid an obstacle). Instead, as discussed above, optical flow makes is less likely to miss actual object movement, so it achieves a higher recall on retaining tracking. It is noteworthy that optical flow estimates object movement in a proactive way, which means the object bounding boxes are shifted according to the pixel motions estimated between the new frame and the previous frame, instead of relying on the motion predictions from the past. It does not pose an assumption on object moving patterns. If the object has appeared in the previous frame, then its movement would be identified. Otherwise, if the object never appears in previous frames, we have a separate module for detecting blobs of new pixels which may correspond to new objects (as will be introduced in Sect. 7.1). Finally, we believe distant new objects that are not easily detected by the blob detection are considered not critical and will be detected in the next full-frame inspection.

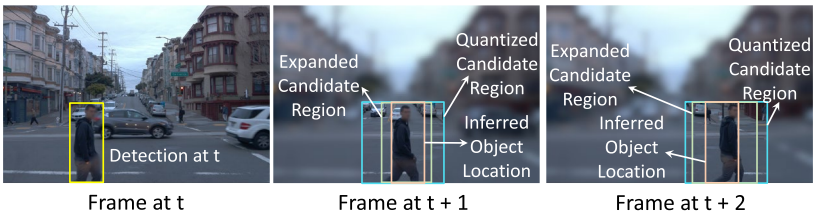


Fig. 2 Comparison between different regions used in the tracking

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

1. *Inferred object location* It tightly bounds the most likely object location as inferred 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 by a vision (AI-based) component.
2. *Expanded candidate region* It expands the inferred object location on account of potential inaccuracy. 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 from frame to frame because each future application of optical flow contrasts the new frame with pixels in the expanded candidate region to produce a new box that encapsulates all new locations of these pixels in the new frame, likely resulting in box enlargement. The effect continues until an inspection of the expanded region is scheduled thus pinpointing the actual object again.
3. *Quantized candidate region* We pad the expanded candidate region to the nearest quantized size from a predefined 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 Sect. 5.

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

4.2.1 Computing inferred 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).

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 := inferred object location
    end
  end
end

4.2.2 Computing expanded candidate regions

This region starts from a previously detected object location, and then 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:

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

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

Thus, we define the new expanded candidate region as:

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

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.

4.2.3 Computing quantized candidate regions

To facilitate batching of image processing, we pad the expanded candidate region to the nearest quantized (squared) *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 three 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 (Torralba 2009).

4.2.4 Data association

After we receive the detected object locations from the detector, we perform data associations between the existing object tracks (represented by their inferred 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 inferred object locations.

4.3 Object location uncertainty

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

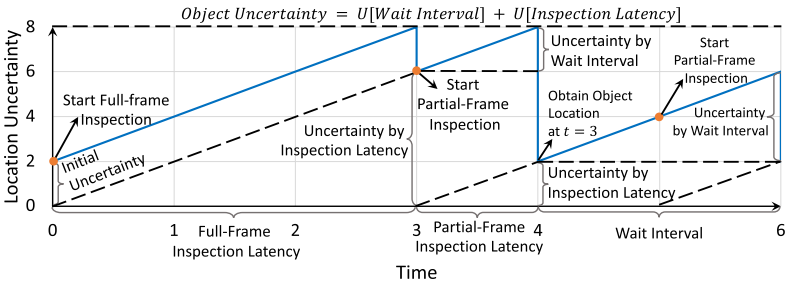


Fig. 3 The object uncertainty comes from both *wait intervals* and *inspection latencies*. The reset value only depends on inspection latency

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. We set object weight as the product of the above two terms to balance the *uncertainty-based prioritization* and the configurable *application-specific criticality assignment*. For example, users can set much higher object criticality to humans than traffic signs such that the inspection frequency of humans is always higher than the traffic signs, no matter their relative moving speed to the ego-vehicle. Instead, setting the same criticality on all objects leads to entirely uncertainty-based prioritization, where object inspection frequencies will be set proportional to their location uncertainty growth rate.

As shown in Fig. 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.

5 Partial-frame inspection scheduling

In this section, we formulate the partial-frame inspection scheduling (PFIS) problem without considering image resizing, and introduce the batched proportional balancing (BPB) algorithm that only applies intermittent inspection and tasking batching, which will be further extended by integrating image resizing in the next section.

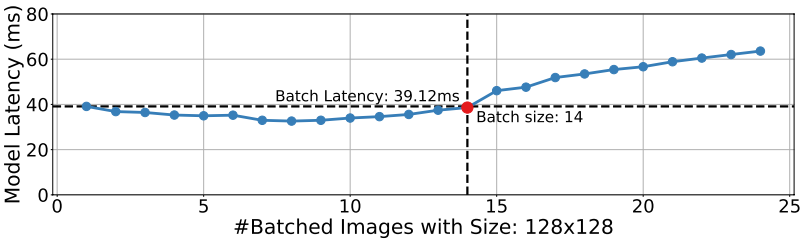


Fig. 4 YOLO execution latencies of 128×128 images with different batch sizes on Jetson Xavier. The inflection point is highlighted in red, where the batch size is 14. We set the batch limit and batch latency correspondingly (Color figure online)

5.1 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 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 Fig. 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 ($1 \leq \kappa \leq \kappa_s$) objects of target size s within time τ_s.

5.2 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 $w_1 \leq \ldots \leq w_N$. 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., $\|\mathcal{N}_j\| \leq \kappa_{s^j}$. (2) We define the *valid period* of a frame as the interval between its arrival and the arrival of the next frame. 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., $t^l + \tau_{s^l} \leq T$.

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.

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.

5.3 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.

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**
  $t_j := \max\{t_j, \text{start of valid period of the } l\text{-th frame}\}$.
  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 $\mathcal{S}$

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 (Holte et al. 1989). 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 subprocedure to derive an optimal schedule for a given inspection frequency set.

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 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,

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

[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 $x_i \leq L$.

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 Fig. 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.

5.4 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: We prove in Lemma 1 that 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: We prove in Lemma 2 that given an inspection frequency set, BAS is optimal in minimizing the overall execution latency.
- Step 3: We prove in Lemma 3 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 in Theorem 1.

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.

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

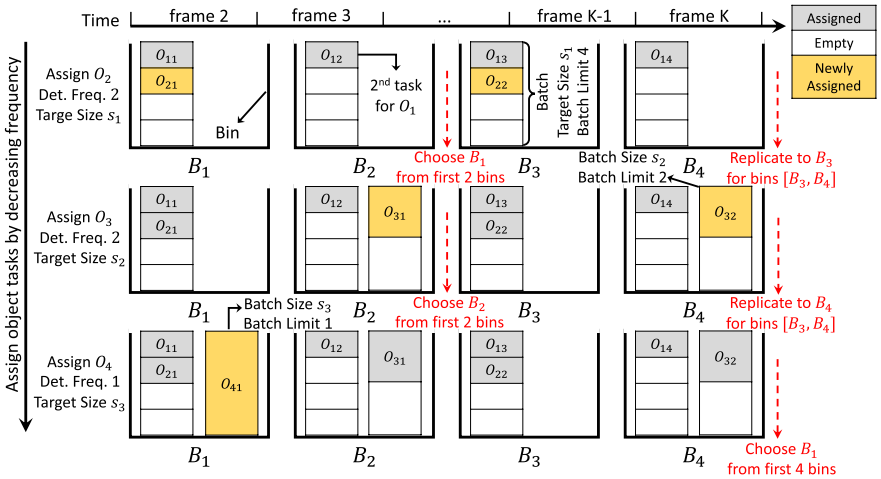


Fig. 5 Graphical illustration on how BAS generates the task-bin mapping. We have four objects denoted by (object, inspection frequency, target size): $(\mathcal{O}_1, 4, s_1)$, $(\mathcal{O}_2, 2, s_1)$, $(\mathcal{O}_3, 2, s_2)$, $(\mathcal{O}_4, 1, s_3)$. We have 4 (virtual) bins, which are **not** 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

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.*

Proof For each i, let $L_i := \frac{L}{x_i}$, where x_i is the detection frequency for object $\mathcal{O}_i$. L_i is an integer power of 2 by construction. When assigning object $\mathcal{O}_i$, we have $\lambda_l(i) = \lambda_{l'}(i)$ for $l \equiv l' (\mod L_i)$ (as exemplified in Fig. 5).

- If there exists an incomplete batch for target size s_i in the first L_i bins, then $\bar{\lambda}(i) = \bar{\lambda}(i-1)$ and $\underline{\lambda}(i) = \underline{\lambda}(i-1)$, and the claim follows.
- If there is no incomplete batch, then based on induction, there are a subset of bins, that for all l such that $\lambda_l(n-1) = \underline{\lambda}(i-1)$, and they are equivalent modulo L_{i-1}. Since $L_{i-1} \leq L_i$ and $L_i \mod L_{i-1} \equiv 0$, there exists at least one bin $l \in L_i$ with load $\underline{\lambda}(i-1)$. After assigning $\mathcal{O}_i$ to l, its load will increase to $\underline{\lambda}(i-1) + \tau_{s_i}$. If $\underline{\lambda}(i-1) + \tau_{s_i} \geq \bar{\lambda}(i-1)$, we have $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \tau_{s_i}$; otherwise, we have $\bar{\lambda}(i) - \underline{\lambda}(i) \leq \bar{\lambda}(i-1) - \underline{\lambda}(i-1)$.

□

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.

Proof As tasks of different sizes cannot be batched, the total load of a schedule is determined by the batching composition of each size. For a given size, the total load of tasks is minimized when the number of batches is minimized. It can be proven by induction that, for the same size, BAS minimizes the number of batches for the first i objects. The base step is trivial, since no tasks of the same object can be batched. For the induction step, assume the claim holds for i. When deciding on the mapping of tasks on object $\mathcal{O}_{i+1}$, if there exists an incomplete batch, then the schedule by BAS has the same number of batches as that for the first i objects, from which the claim follows. If no incomplete batch exists, consider any partial schedule for the first i objects. If for that partial schedule, an incomplete batch exists, then it contains at least one more batch than the partial schedule by BAS for the first i objects. It is contradictory with the assumption that BAS minimizes the batch count for the first i objects. In both cases, for the first $i+1$ objects, the partial schedule by BAS contains no more batches than any other partial schedule, which completes the induction argument. □

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. 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 $\tilde{x}_i^* \geq \frac{w_i \tilde{x}_{\hat{i}}^*}{w_{\hat{i}}} \geq \left\lfloor \frac{w_i \tilde{x}_i^*}{w_{\hat{i}}} \right\rfloor$. 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 $\mathcal{O}_i$,

[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 i with non-zero value of this equation and leave out the remaining objects.

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

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

We have proved (Lemma 2) that BAS minimizes the system load, thus the factor c by BAS is at least $\hat{c}$, i.e., $c \ge \hat{c}$. 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 $\bar{\lambda}(N) \le 2\underline{\lambda}(N)$, we have

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

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

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

(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 $\bar{\lambda}(N) - \underline{\lambda}(N) \le \bar{\lambda}(i) - \underline{\lambda}(i) \le \tau_{s_i}$. We have $\tau_{s_i} \ge \max_l \frac{\lambda_l}{2}$. Even under the optimal schedule, the maximum uncertainty of object $\mathcal{O}_1$ is at least $w_1\tau_{s_i} \le \tilde{U}^*$, so we have $\max_l \lambda_l \le \frac{2\tilde{U}^*}{w_1}$. Hence,

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

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 $\tilde{U} \le \tilde{U}^*$ in both cases. □

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 $\tilde{U}^* \le U^*$. They can correspond to two different schedules. We have,

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

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. □

6 Generalized scheduling with image resizing

In previous formulation, we established the scheduling problem as a max-uncertainty minimization problem by controlling the object inspection frequency along with their batching decisions. In this section, we generalize the scheduling problem by further considering image resizing. We first introduce the design rationale behind integrating image resizing, followed by a formulation of the generalized scheduling algorithm, and finally explain how we extend the proposed scheduling algorithm to solve it.

6.1 Design rationale

As addressed before, common CNN-based object detection models, like YOLO, are able to process input images of different resolutions with different latencies, as long as they are subject to certain constraints (e.g., the image sizes should be multiples of 32 in YOLO). To utilize this property, we can not only slice out partial regions of different sizes, but also resize the large regions into smaller sizes. Reducing the image resolutions (or called *image resizing*) before feeding them into DNN models trades part of the model accuracy for reduced execution latency. We investigate the impact of image resizing on achieved detection quality and inference latency. We downsize the full images to different resolutions, and evaluate the detection accuracy (i.e., mAP[5]) on objects of different sizes, as well as the associated inference latency. As shown in Fig. 6, image resizing leads to different accuracy degradation curves on objects of different sizes. Large objects are more tolerable to image resizing than small objects, and very large objects (object size $\geq$ 160) almost suffer from no accuracy degradation even if we reduce the image size by 5 times (i.e., downsize ratio as 0.2). According to our profiling result, image resizing runs fast on Jetson Xavier with a latency of <1ms in most cases. Besides, as we will introduce in Sect. 7.2, we merge highly overlapped partial regions, so only a limited number of large partial regions are extracted from each frame.

There are two potential benefits if we integrate image resizing into the proposed scheduling framework. First, after downsizing some partial frames, the execution latency of every single inspection will decrease, thus we can schedule more inspections for both downsized objects and the remaining objects. Second, image resizing could further facilitate task batching of partial frames which originally have different sizes. For example, if we downsize partial frames of 512×512 to 256×256, we can further batch them with the remaining 256×256 partial frames. Ultimately,

[5] The specific definition of the metric will be given later.

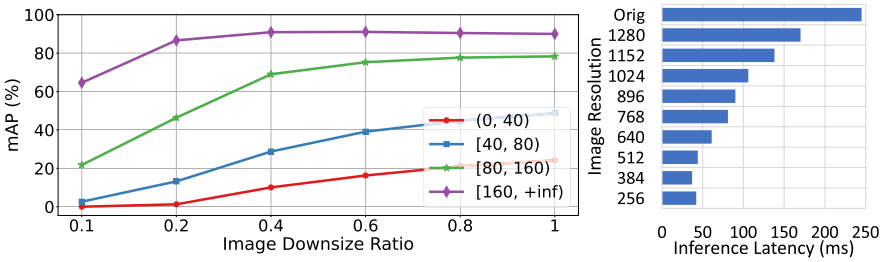


Fig. 6 Impact of image resizing on detection quality and inference latency. In the left figure, different curves represent detection accuracy degradation on objects of different sizes

lower system uncertainty can be achieved with the appropriate use of image resizing. However, we can not ignore the detection quality degradation caused by image resizing, which is not reflected in system uncertainty, thus it can no longer be used as a proper proxy objective of the object detection accuracy. We next solve this issue by defining a generalized optimization objective and scheduling algorithm.

6.2 Generalized scheduling problem formulation

To address the aforementioned issue, we first define the execution model related to image resizing operation, and then define a new optimization objective, named *generalized uncertainty*, integrating the effect of image downsizing, inspection frequency, and task batching.

6.2.1 Image resizing and accuracy degradation

As defined before, each object $\mathcal{O}_i$ is associated with a fixed target size $s_i \in \{s_1, \ldots, s_M\}$ within a scheduling horizon, which was calculated by expanding and quantizing the inferred object location at the first frame after the full frame inspection. It represents the size of the partial frame to be processed in each of its inspection task. We further define a *confined target size* for each object below.

Definition 4 (*Confined target size*) The confined target size $s'_i \in \{s_1, \ldots, s_M\}$ for object $\mathcal{O}_i$ could be any size in the limited size set that is no larger than s_i, i.e., $s'_i \leq s_i$. To maintain the task batching opportunities, the same confined target size for an object persists across a scheduling horizon. At one frame, if the expanded candidate region is larger than the confined target size, we would first pad the expanded candidate region into a square region (to keep the same aspect ratio), and then downsize it to s'_i.

In order to depict the accuracy degradation associated with image resizing, we define an *accuracy degradation profile*, which is essentially a lookup table using the object size[6] and downsizing ratio as two input dimensions (as shown in the left

[6] We want to remind that the object size is not identical to the object target size because the target size not only depends on the object size, but also the object motion.

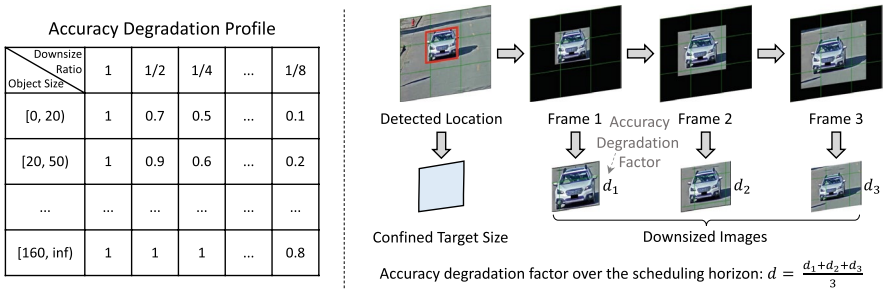


Fig. 7 Illustration of the accuracy degradation profile and degradation factor

part of Fig. 7). The element in each cell represents the ratio between the accuracy on downsized images and original images, which is also called the *accuracy degradation factor*. This table is obtained through offline profiling by downsizing the images with different downsize ratios and separately counting the obtained accuracy on different object sizes.

To accurately calculate the accuracy degradation factor $d_i(t)$ of object $\mathcal{O}_i$ at time t, the following steps should be performed: We first compute the size of its expanded candidate region according to its uncertainty growth rate u_i and the time since its last inspection $t - t_i$. Next, we get the downsize ratio by calculating the ratio between its confined target size and expanded candidate region size. Finally, we use the detected object size and the downsize ratio to query the accuracy degradation profile to get its accuracy degradation factor.

Since the object region projections are iteratively expanded in each frame, with a given confined target size, the downsize ratio of an object is different at each frame. As a consequence, the exact accuracy degradation of an object over a scheduling horizon also depends on its inspection schedule (i.e., at which frame the inspection is performed). This could significantly complicate the scheduling problem since we use the accuracy degradation profile as input to the scheduling algorithm. To simplify the problem, we propose an approximated approach to calculate the accuracy degradation factor of an object over a scheduling horizon in a schedule-independent way, which is defined below.

Definition 5 (*Approximated object accuracy degradation factor in a scheduling horizon*) Given an object $\mathcal{O}_i$, and its calculated accuracy degradation factor $d_i(1), \ldots, d_i(T-1)$ at each frame (when no partial frame inspection is performed), its approximated accuracy degradation factor for the whole scheduling horizon is calculated as the weighted average of its accuracy degradation factors at each frame, discounting the weights of frames by their distance to the key frame,

$$d_i = \frac{d_i(1) + \frac{1}{2}d_i(2) + \cdots + \frac{1}{2^{T-2}}d_i(T-1)}{1 + \frac{1}{2} + \cdots + \frac{1}{2^{T-2}}}. \tag{2}$$

There are two reasons behind this design choice: First, most objects are more likely to get high inspection frequencies than low inspection frequencies. Second, even if we assume the same probability for different inspection frequencies, high inspection frequencies will lead to higher absolute inspection times, which happen at frames close to last inspected frames. In such a way, we formulate the impact of image resizing independent of the specific object inspection schedules, and leave the image resizing a higher level decision above the inspection frequency.

6.2.2 Generalized problem formulation

In the generalized problem formulation, we intend to integrate the effect of object location uncertainty caused by skipped object inspection and the detection accuracy degradation caused by image resizing defined above. We first define the generalized weighted object uncertainty $GU_i(t)$ at time t as,

Definition 6 (*Generalized object uncertainty*) Given an object $\mathcal{O}_i$ and the last inspection time t_i, its generalized uncertainty $GU_i(t)$ at time t is defined as the ratio between its location uncertainty and its approximated accuracy degradation factor in the scheduling horizon:

$$GU_i(t) = \frac{U_i(t)}{d_i} = \frac{w_i(t - t_i) + u_i}{d_i}. \tag{3}$$

Intuitively, we should compensate the downsized objects with more inspections to bound its generalized uncertainty, otherwise we will be more likely to lose the tracking of this object. Next, the generalized partial-frame inspection scheduling problem is correspondingly defined as,

Definition 7 (*Generalized partial-frame inspection scheduling problem*) The Generalized Partial-Frame Inspection Scheduling (GPFIS) problem asks for a feasible schedule that minimizes the system uncertainty within a scheduling horizon.

The definition of schedule remains the same as Definition 1 except that we also need to decide a confined target size for each object. We next introduce how can we extend the proposed BPB policy to solve the new GPFIS problem.

6.3 Generalized BPB scheduling policy

There is a tradeoff between the decision space complexity and scheduling flexibility when we jointly determine the downsizing factors for all objects. If we determine a confined target size for all objects, the decision space is constant to the number of objects, but the achieved model accuracy could be suboptimal. On the contrary, if we separately determine a confined target size for each object, the achieved model accuracy is optimized, but the decision space grows exponentially to the number of objects, which makes the problem NP-hard even if we do not consider the inspection

schedule. Therefore, we propose to decide the resizing factors at the granularity of object groups segmented by the object sizes, because it has been shown in Fig. 6 that the difficulty of detecting objects with different sizes are different with heterogeneous accuracy degradation curves. Large objects are more tolerable to image resizing, while small objects can not be aggressively downsized.

Even under this restricted setting, the decision choices is still exponential to the number of (discretized) object sizes. As we can observe from Fig. 6, it is always more beneficial to downsize larger objects than smaller objects, because doing so saves more in DNN model latency (with more inspection area reduction), while suffering from less accuracy degradation, under the same downsizing factor. Thus, we further reduce the decision space by only deciding a maximum target size $s_{\max} \in \{s_1, \dots, s_M\}$ such that any larger candidate regions should be downsized to s_{max}.

Specifically, we propose the *Generalized Batched Proportional Balancing (GBPB)* policy in Algorithm 4, which uses Algorithm 2 as a subroutine, and feed different max target size values to compute the optimized schedule that achieves the minimum generalized system uncertainty. We use a loop to iterate over all used target sizes (too large target sizes are skipped if no object uses them). At each iteration, we first use the selected max target size to calculate the confined target sizes $\{s'_1, \dots, s'_N\}$ and the approximated object accuracy degradation factors $\{d_1, \dots, d_N\}$ for the objects, and correspondingly update the object weights to $\{\frac{w_1}{d_1}, \dots, \frac{w_N}{d_N}\}$. Next, they are fed into Algorithm 2 to compute the object inspection schedule and the achieved generalized system uncertainty. The maximum target size value that achieves the minimum generalized system uncertainty and its schedule are used as the algorithm output. Since we greatly reduce the decision space of image resizing and decouple the impact of image resizing and intermittent inspection, Algorithm 2 will be called at most M times (i.e. the number of target sizes) in actual scheduling execution.

Algorithm 4: Generalized Batched Proportional Balancing (GBPB) Policy

Input: Object set $\{\mathcal{O}_1, \dots, \mathcal{O}_N\}$, object weights $\{w_1, \dots, w_N\}$, number of frames $K-1$, target size set $\{s_1, \dots, s_M\}$.
Output: A feasible schedule with minimized generalized uncertainty.
Initialize: Min uncertainty $GU_{\min} := +\infty$, output schedule $\mathcal{S}_{out}$;
for $s_{\max} \in \{s, \dots, s_2, s_1\}$ *(reverse order of used target sizes)* **do**
  Calculate the confined target sizes $\{s'_1, \dots, s'_N\}$ and the approx. acc. degradation factors $\{d_1, \dots, d_N\}$ with $s_{\max}$;
  Update the object weights as $\{\frac{w_1}{d_1}, \dots, \frac{w_N}{d_N}\}$;
  Feed $\{s'_1, \dots, s'_N\}$ and $\{\frac{w_1}{d_1}, \dots, \frac{w_N}{d_N}\}$ into **Algorithm 2**, return the achieved generalized uncertainty GU and schedule $\mathcal{S}$ (including the confined target sizes);
  if $GU < GU_{\min}$ **then**
    $GU_{\min} := GU,\ S_{out} := S$;
  end
end
Return the schedule S_{out}.

7 Empirical optimization

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

7.1 New object arrival

We first show in Fig. 8 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 (Grana et al. 2010) to extract new object regions. An example is shown in Fig. 9.

7.2 Partial region merge

If two candidate regions have significant overlap, 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 such that its overlap ratio with a scheduled region is above a threshold I, we use the merged region to replace the scheduled region. This approach could help reduce the redundant inspections of overlapping regions on different partial frames extracted from the same frame.

7.3 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 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 Fig. 10. Intact redundant inspections can be easily removed by the non-maximum suppression (NMS) step of the detector.

8 Evaluation

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

8.1 Experimental setup

8.1.1 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 with the maximum CPU/GPU/memory frequency capacity.

8.1.2 Dataset

Our experiment is performed on the Waymo Open Dataset (Sun et al. 2020), a large-scale autonomous driving dataset collected by Waymo self-driving cars in diverse geographies and conditions. It consists of driving video segments of 20 s each, collected by onboard cameras at 10Hz with resolution 1920×1280. Only front camera data is used.

8.1.3 Neural network for detection

We use the YOLOv5[7] model in PyTorch as the object detection network, which was pretrained on the general-purpose COCO (Lin et al. 2014) 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 latencies with different target sizes are profiled in advance.

8.1.4 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 (150 ms, 100 ms, and 70 ms) to denote the easy, moderate, and hard scheduling situations (corresponding to frame rates of roughly 6.67 Hz, 10 Hz, and 14 Hz).

8.1.5 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 objects. 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

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

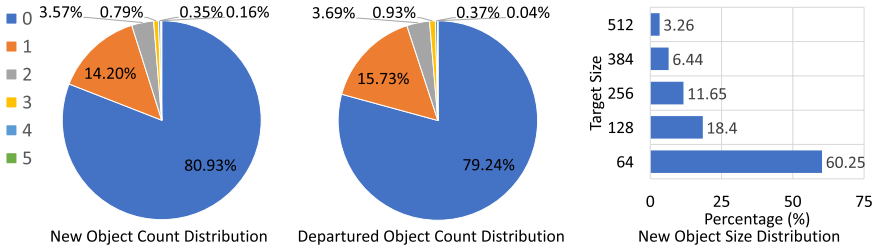


Fig. 8 Distributions on number of *newly arrived objects* and *departured objects*, as well as the *(quantized) new object size distribution*, at each frame. Results obtained on Waymo Open dataset (Sun et al. 2020)

Fig. 9 An example of new pixels in the current frame highlighted in red (Color figure online)

performance on *all objects* and *critical objects*. A separate object size threshold for critical objects is set for each class.

8.1.6 Evaluation metrics

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 overlap. 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) and larger than the remaining detections, in which case we say that the object is *successfully detected*. It has been shown by Liu et al. (2022b) that the achieved localization error and classification accuracy are similar across different approaches, so they are skipped in this evaluation. The following set of metrics are utilized:

- 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.
- 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[8] is used.

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

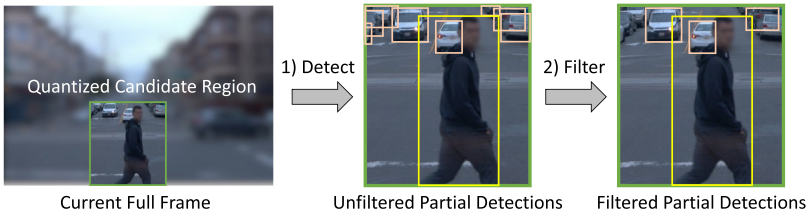


Fig. 10 An example of bounding box filtering. Note we preserve the yellow box although its bottom edge lies on the bottom border of the partial image, because the partial image bottom coincides with the full image bottom (Color figure online)

The YOLO performance on full frames is listed in Table 1, which serves as the ceiling condition for the proposed framework. In addition, we separately evaluate the detection performance on all objects and the critical objects.

8.2 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 and without bounding box filtering, under different candidate region merge criteria (i.e. the intersection ratios) in Fig. 11. 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 Fig. 11b 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.

8.3 Tracking algorithm

We compare our flow-based tracker (denoted by "`Flow`") with a state-of-the-art tracking algorithm, `SORT` (Bewley et al. 2016), which uses a Kalman filter to model object motions. It extrapolates future object locations from the past object trajectories. The results are presented in Fig. 12. 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`.

8.4 Robustness of flow-based tracking and slicing

In this experiment, we explicitly evaluate the robustness of the proposed flow-based tracking algorithm, by answering 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 Fig. 13. We use a 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.

8.5 Scheduling algorithm comparison

8.5.1 Baselines

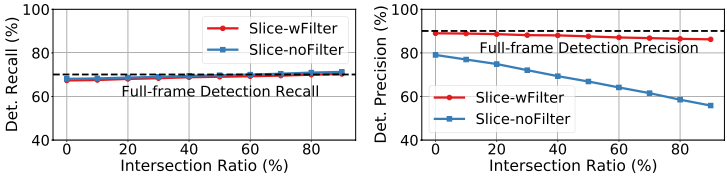
We compare the following algorithms in this evaluation.

- 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) (Liu et al. 2020a): 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.
- Batched proportional balancing policy (BPB) (Liu et al. 2022b): The proposed scheduling algorithm in Sect. 5 that controls the inspection frequency of each

Table 1 YOLOv5 performance on Waymo dataset

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	XavierLatency	
	62.76	78.14	239ms	

All values in this table are in percent, except the latency



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

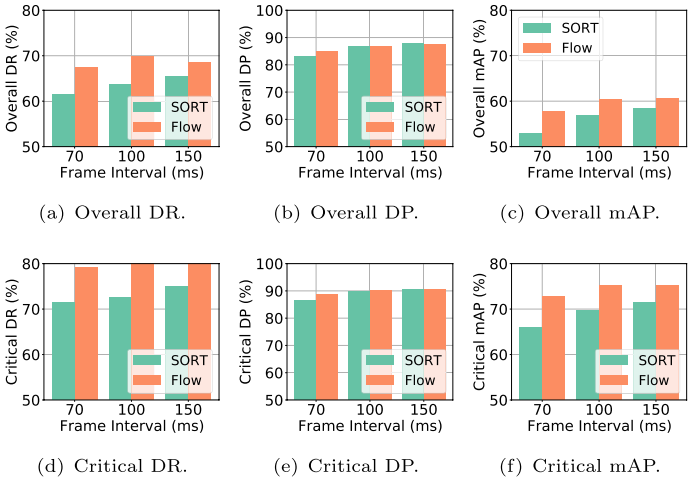
Fig. 11 Impact of slicing on detection quality

object according to their criticality and uncertainty growth. No image resizing is applied.

- Generalized batched proportional balancing policy (GBPB): The generalized scheduling algorithm in Sect. 6 that integrates image resizing and intermittent inspection, as well as task batching in the optimization.

8.5.2 Results

The corresponding results are summarized in Fig. 14. We test the scheduling algorithms at different workloads (i.e. frame intervals) and report the following observations: First, both BPB and GBPB improve the mAP compared to the baselines, a metric that captures both localization and classification errors in object detection. GBPB further improves the mAP of BPB by introducing image resizing in the scheduling framework, especially when the frame interval is short (i.e. 70 ms). Instead, BPB and GBPB achieve similar mAP when the frame interval is 150ms, proving that GBPB can maximally utilize the available resources by setting larger max confined target sizes when the computation demand is low. Through the comparison, we conclude that GBPB achieves better computational resiliency than BPB with less degradation in the face of higher workloads (i.e. 70 ms). This is because image resizing provides an alternative dimension for computational savings, which avoids extremely low inspection frequency on critical objects. Otherwise, relying too much on optical flow-based projection to predict object locations impairs tracking of fast-moving objects. Second, GBPB achieves the best recall, but at the cost



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

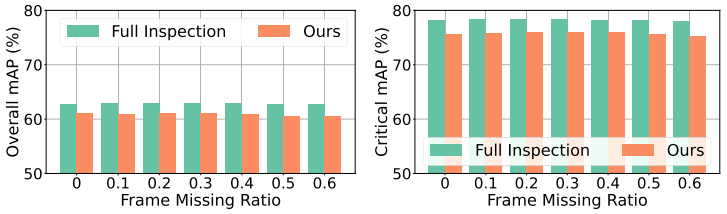
(d) Critical DR. (e) Critical DP. (f) Critical mAP.

Fig. 12 The impact of tracking algorithms on the detection quality of overall objects and critical objects

of minor degradation in precision, compared to full-frame downsizing (DS). This is because both the image slicing and partial-frame downsizing can result in minor degradation in the detection precision, which is less important than detection recall. Finally, compared to BHUF, the proposed approaches do better at planning when to invoke the detector to minimize (generalized) location error without hurting recall.

8.5.3 Evaluation on busy traffic scenarios

To further demonstrate the additional value of GBPB over BPB, we separately extract a new dataset from Waymo with dense object distributions, which are not in the previous dataset, to evaluate the performance of the new GBPB algorithm under busy traffic scenarios. Example scenarios of this dataset are visualized in Fig. 15 and the associated detection results are presented in Fig. 16. We find the advantage of GBPB over BPB is larger in busy traffic scenarios, which achieves 5.42% overall mAP and 6.33% critical mAP improvement when the frame interval is 70 ms. When the number of objects in the scene is large, BPB can only reduce the inspection frequency of all objects, which leads to more "inferred object locations from the past" in the generated detections that are possibly shifted from the actual object locations, while GBPB can balance between lower inspection frequencies and lower image resolutions, which offers a larger optimization space in combination. We also notice that the image downsizing in GBPB leads to minor degradation in the achieved detection precision (DP), in exchange for higher detection recall (DR), but GBPB still outperforms BPB on the mAP metric. The results in this experiment also validate our previous observation that GBPB provides better resiliency on machine perception in facing higher computation workload through applying its



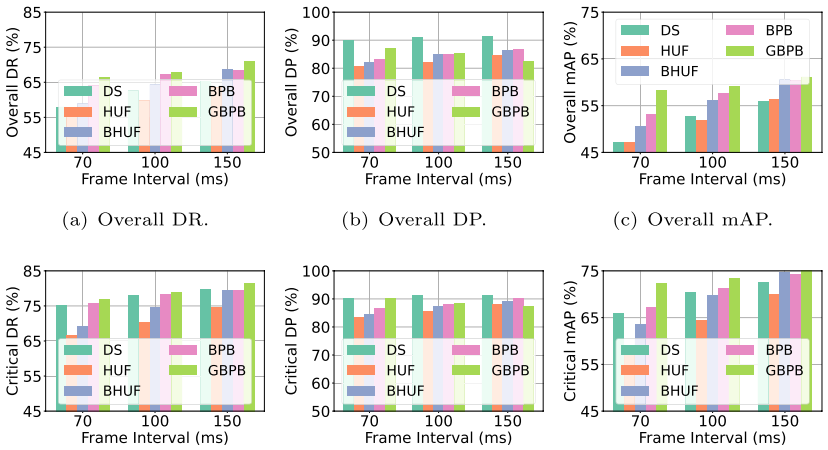
(a) Overall mAP. (b) Critical mAP.

Fig. 13 The impact of missing frames on our slicing and partial inspection approach

multi-dimensional resource allocation, which slightly reduces the inspection fidelity on downsized images, but sustains a reasonable range of inspection frequencies.

8.6 Responsiveness to physically close objects

We evaluate the achieved detection recall on all objects within 30 ms to the ego-vehicle (according to the groundtruth object distances in the dataset). The results are normalized by the detection recall achieved on full frames, and reported in Fig. 17. Both BPB and GBPB outperform the baselines in close object recall, 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. We also acknowledge that GBPB shows slightly worse recall on

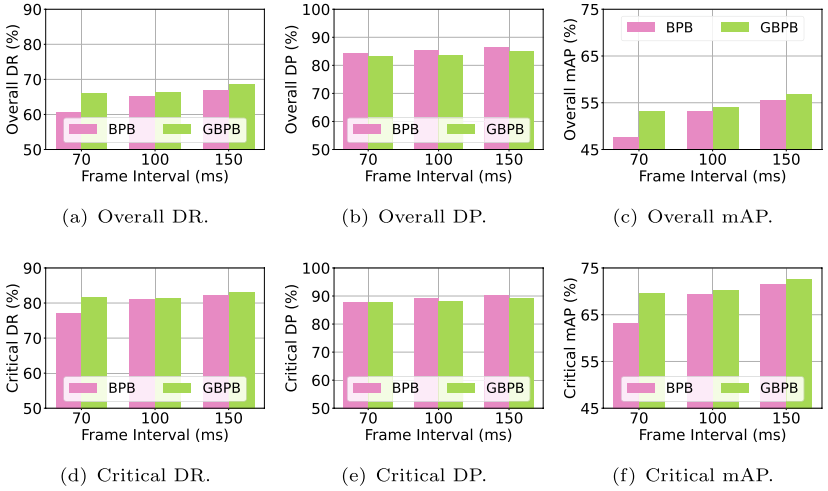


(a) Overall DR. (b) Overall DP. (c) Overall mAP.

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

Fig. 14 Scheduling algorithms comparison. The first row shows detection results on overall objects, and the second row shows results on critical objects

Fig. 15 Example scenarios in the extracted busy-traffic dataset



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

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

Fig. 16 Scheduling algorithms comparison on busy traffic scenarios. The first row shows detection results on overall objects, and the second row shows results on critical objects

close objects than BPB, because we only differentiate the object resizing decisions according to their sizes, but do not consider their criticality.

8.7 Breakdown of overhead quantification

Next, we separately report the breakdown latency overhead induced by BPB and GBPB algorithms. The results are shown in Fig. 18. Since the BPB and GBPB policy only executes once per scheduling horizon, their overhead is divided into each frame. We notice that the per-frame latency of GBPB increases to < 9 ms compared to < 2 ms in BPB, which is because we iterate over different max confined target sizes in Algorithm 4 by repetitively calling Algorithm 2 as a subrountine. Since different max confined target sizes are independently tested, we can further improve the efficiency of GBPB by scheduling their Algorithm 2 calls in parallel on different CPUs in the future. The preprocessing overhead is generally below 20 ms and the postprocessing overhead is generally below 5 ms in both algorithms, which

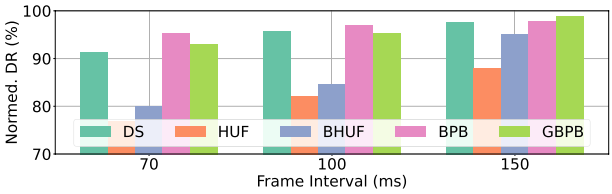
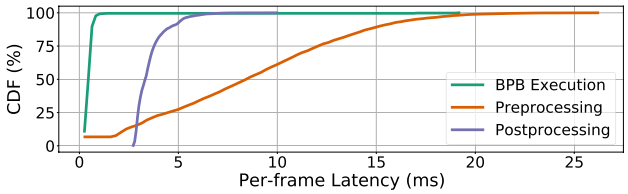
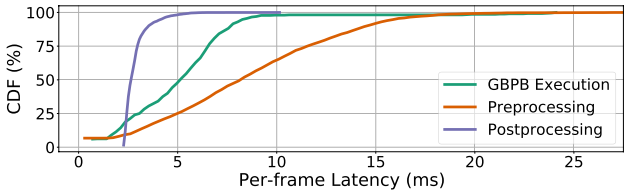


Fig. 17 The normalized detection recall on physically close objects



(a) BPB Overhead Distribution.



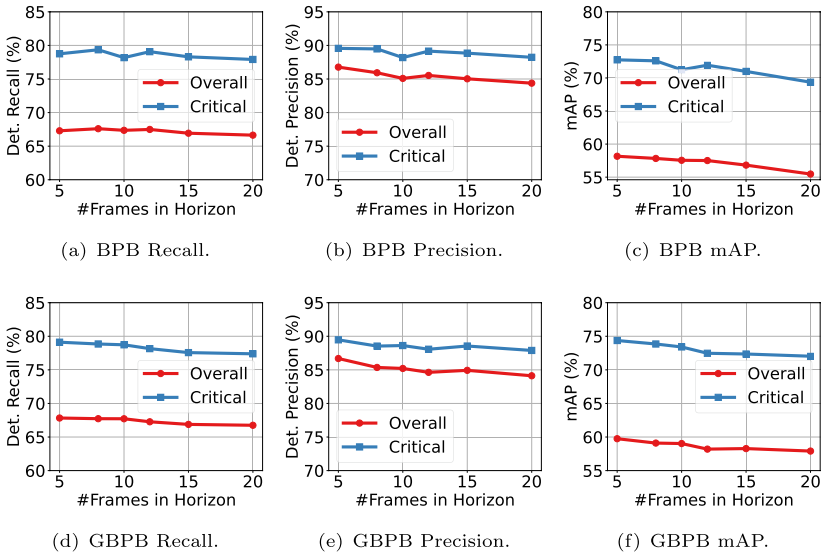
(b) GBPB Overhead Distribution.

Fig. 18 Breakdown latency overhead of the proposed algorithms

are acceptable compared to 100 ms frame intervals. Specifically, the preprocessing steps include new object localization, image slicing, candidate region merge, resizing, 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. In conclusion, the proposed scheduling framework does not induce significant latency overhead (generally below 30 ms) to the backbone DNN inspection system on Jetson Xavier.

8.8 Choice of scheduling horizon length

Here, we investigate the impact of the scheduling horizon length in BPB and GBPB policies. We vary the horizon length from 5 to 20 frames, and evaluate how the detection quality is affected. We set the frame arrival interval $P = 100$ ms. The results are summarized in Fig. 19. Both BPB and GBPB are generally resilient to the horizon length, and large variations on achieved object detection quality are not



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

(d) GBPB Recall. (e) GBPB Precision. (f) GBPB mAP.

Fig. 19 The impact of the scheduling horizon length on detection quality

seen. Relatively short scheduling horizons show slightly better mAP than longer scheduling horizons. 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. However, when the scheduling horizons are too short, the full-frame inspections would be frequently invoked, which could lead to delays in obtaining object locations in real time. Therefore, we believe a moderate length of scheduling horizon (i.e. 10 frames) is the best choice.

9 Conclusions

We proposed a generalized self-cueing attention scheduling framework that integrates intermittent inspection, image resizing, and task batching to optimize the efficiency of DNN-based visual machine perception pipelines on resourced-constrained embedded platforms. It minimizes a concept of generalized system uncertainty that simultaneously considers the detection accuracy degradation caused by image resizing and object location uncertainty caused by skipped object inspections. Under the time constraint, the proposed scheduling algorithm, called GBPB policy, can balance between the inspection quality and the inspection frequencies on object regions depending on the object motion distributions and object size distributions. Extensive evaluations with real-world driving datasets on an NVIDIA Jetson Xavier platform demonstrate the effectiveness of the proposed scheduling framework. Compared with the original BPB policy in Liu et al. (2022b), the additional image resizing component in GBPB is especially effective when dealing with busy traffic scenarios under short time limits because resizing offers a better alternative to extreme reduction in inspection frequency, especially in the presence of large objects.

Acknowledgements Research reported in this paper was sponsored in part by the U.S. DEVCOM Army Research Laboratory under Cooperative Agreement W911NF-17-20196, NSF CNS 20-38817, IBM (IIDAI), and the Boeing Company. 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 U.S. DEVCOM Army Research Laboratory or the U.S. government. The U.S. government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

References

Amert T, Otterness N, Yang M, et al (2017) GPU scheduling on the nvidia tx2: hidden details revealed. In: 2017 IEEE real-time systems symposium (RTSS), IEEE, pp 104–115

Amert T, Tong Z, Voronov S, et al (2021) Timewall: enabling time partitioning for real-time multicore+ accelerator platforms. In: 2021 IEEE real-time systems symposium (RTSS), IEEE, pp 455–468

Bastani F, Madden S (2021) Multiscope: efficient video pre-processing for exploratory video analytics. CoRR abs/2103.14695. arXiv:2103.14695

Bateni S, Liu C (2018) Apnet: approximation-aware real-time neural network. In: 2018 IEEE real-time systems symposium (RTSS), IEEE, pp 67–79

Bewley A, Ge Z, Ott L, et al (2016) Simple online and realtime tracking. In: 2016 IEEE international conference on image processing (ICIP), IEEE, pp 3464–3468

Buckler M, Bedoukian P, Jayasuriya S, et al (2018) Eva^2: Exploiting temporal redundancy in live computer vision. In: 2018 ACM/IEEE 45th annual international symposium on computer architecture (ISCA), IEEE, pp 533–546

Capodieci N, Cavicchioli R, Bertogna M, et al (2018) Deadline-based scheduling for gpu with preemption support. In: 2018 IEEE real-time systems symposium (RTSS), IEEE, pp 119–130

Cavigelli L, Degen P, Benini L (2017) Cbinfer: change-based inference for convolutional neural networks on video data. In: Proceedings of the 11th international conference on distributed smart cameras, pp 1–8

Chin T, Ding R, Marculescu D (2019) Adascale: Towards real-time video object detection using adaptive scaling. In: Talwalkar A, Smith V, Zaharia M (eds) Proceedings of machine learning and systems 2019, MLSys 2019, Stanford, CA, USA, March 31–April 2, 2019. mlsys.org

Grana C, Borghesani D, Cucchiara R (2010) Optimized block-based connected components labeling with decision trees. IEEE Trans Image Process 19(6):1596–1609

Heo S, Cho S, Kim Y, et al (2020) Real-time object detection system with multi-path neural networks. In: 2020 IEEE real-time and embedded technology and applications symposium (RTAS), IEEE, pp 174–187

Heo S, Jeong S, Kim H (2022) Rtscale: Sensitivity-aware adaptive image scaling for real-time object detection. In: 34th euromicro conference on real-time systems (ECRTS 2022), Schloss Dagstuhl-Leibniz-Zentrum für Informatik

Holte R, Mok A, Rosier L, et al (1989) The pinwheel: A real-time scheduling problem. In: Proceedings of the 22nd Hawaii international conference of system science, pp 693–702

Hu Y, Liu S, Abdelzaher T, et al (2021) 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, pp 169–178

Hu Y, Liu S, Abdelzaher T, et al (2022) Real-time task scheduling with image resizing for criticality-based machine perception. Real-Time Systems pp 1–26

Jang W, Jeong H, Kang K, et al (2020) R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving. In: In Proc. IEEE Real-time Systems Symposium (RTSS)

Ji M, Yi S, Koo C, et al (2022) Demand layering for real-time dnn inference with minimized memory usage. In: 2022 IEEE real-time systems symposium (RTSS), IEEE, pp 291–304

Kang W, Lee K, Lee J, et al (2021) Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks. In: 2021 IEEE real-time systems symposium (RTSS), IEEE, pp 329–341

Kang D, Lee S, Chwa HS, et al (2022a) Rt-mot: Confidence-aware real-time scheduling framework for multi-object tracking tasks. In: 2022 IEEE real-time systems symposium (RTSS), IEEE, pp 318–330

Kang W, Chung S, Kim JY, et al (2022b) Dnn-sam: Split-and-merge dnn execution for real-time object detection. In: 2022 IEEE 28th real-time and embedded technology and applications symposium (RTAS), IEEE, pp 160–172

Kannan T, Hoffmann H (2021) Budget rnns: Multi-capacity neural networks to improve in-sensor inference under energy budgets. In: 2021 IEEE 27th real-time and embedded technology and applications symposium (RTAS), IEEE, pp 143–156

Kroeger T, Timofte R, Dai D, et al (2016) Fast optical flow using dense inverse search. In: European conference on computer vision, Springer, pp 471–488

Kumar AR, Ravindran B, Raghunathan A (2019) 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, pp 150–156

Lee S, Nirjon S (2020a) 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, pp 175–190

Lee S, Nirjon S (2020b) 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, pp 15–29

Li X, Yin F, Zhang X, et al (2021) Adaptive scaling for archival table structure recognition. In: Lladós J, Lopresti D, Uchida S (eds) 16th International Conference on Document Analysis and Recognition, ICDAR 2021, Lausanne, Switzerland, September 5-10, 2021, Proceedings, Part I, Lecture Notes in Computer Science, vol 12821. Springer, pp 80–95

Lin TY, Maire M, Belongie S, et al (2014) Microsoft coco: Common objects in context. In: European conference on computer vision, Springer, pp 740–755

Liu S, Yao S, Fu X, et al (2020a) On removing algorithmic priority inversion from mission-critical machine inference pipelines. In: In Proc. IEEE real-time systems symposium (RTSS)

Liu S, Yao S, Li J et al (2020) Giobalfusion: a global attentional deep learning framework for multisensor information fusion. Proc ACM Interactive Mob Wearable Ubiquitous Technol 4(1):1–27

Liu S, Yao S, Fu X, et al (2021) Real-time task scheduling for machine perception in intelligent cyber-physical systems. IEEE Trans Comput

Liu L, Dong Z, Wang Y, et al (2022a) Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles. In: 2022 IEEE real-time systems symposium (RTSS), IEEE, pp 305–317

Liu S, Fu X, Wigness M, et al (2022b) Self-cueing real-time attention scheduling in criticality-aware visual machine perception. In: Proceedings of the 28th IEEE real-time and embedded technology and applications symposium (RTAS)

Liu S, Wang T, Guo H, et al (2022c) Multi-view scheduling of onboard live video analytics to minimize frame processing latency. In: 2022 IEEE 42nd international conference on distributed computing systems (ICDCS), pp 503–514

Liu S, Wang T, Li J, et al (2022d) Adamask: Enabling machine-centric video streaming with adaptive frame masking for dnn inference offloading. In: Proceedings of the 30th ACM international conference on multimedia, pp 3035–3044

Mao H, Kong T, Dally WJ (2018) Catdet: cascaded tracked detector for efficient object detection from video. arXiv:1810.00434

Minnehan B, Savakis A (2019) Cascaded projection: End-to-end network compression and acceleration. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 10,715–10,724

Najibi M, Singh B, Davis L (2019) Autofocus: Efficient multi-scale inference. In: 2019 IEEE/CVF international conference on computer vision, ICCV 2019, Seoul, Korea (South), October 27–November 2, 2019. IEEE, pp 9744–9754

Razavi K, Luthra M, Koldehofe B, et al (2022) Fa2: fast, accurate autoscaling for serving deep learning inference with sla guarantees. In: 2022 IEEE 28th real-time and embedded technology and applications symposium (RTAS), IEEE, pp 146–159

Redmon J, Divvala S, Girshick R, et al (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788

Restuccia F, Biondi A (2021) Time-predictable acceleration of deep neural networks on FPGA SOC platforms. In: 2021 IEEE real-time systems symposium (RTSS), IEEE, pp 441–454

Song Z, Fu B, Wu F, et al (2020) Drq: dynamic region-based quantization for deep neural network acceleration. In: 2020 ACM/IEEE 47th annual international symposium on computer architecture (ISCA), IEEE, pp 1010–1021

Soyyigit A, Yao S, Yun H (2022) Anytime-lidar: deadline-aware 3D object detection. In: 2022 IEEE 28th international conference on embedded and real-time computing systems and applications (RTCSA), IEEE, pp 31–40

Sun P, Kretzschmar H, Dotiwalla X, et al (2020) Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 2446–2454

Torralba A (2009) How many pixels make an image? Vis Neurosci 26(1):123–131

Wang S, Lu H, Deng Z (2019) Fast object detection in compressed video. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 7104–7113

Wu J, Subasharan V, Tran T, et al (2022) MRIM: enabling mixed-resolution imaging for low-power pervasive vision tasks. In: IEEE international conference on pervasive computing and communications, PerCom 2022, Pisa, Italy, March 21–25, 2022. IEEE, pp 44–53

Xiang Y, Kim H (2019) Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In: 2019 IEEE real-time systems symposium (RTSS), IEEE, pp 392–405

Xu M, Zhu M, Liu Y, et al (2018) Deepcache: principled cache for mobile deep vision. In: Proceedings of the 24th annual international conference on mobile computing and networking, pp 129–144

Yang Z, Nahrstedt K, Guo H, et al (2021) Deeprt: a soft real time scheduler for computer vision applications on the edge. arXiv:2105.01803

Yao S, Zhao Y, Shao H, et al (2018) 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, pp 278–291

Yao S, Hao Y, Zhao Y, et al (2020a) Scheduling real-time deep learning services as imprecise computations. In: Proc. IEEE international conference on embedded and real-time computing systems and applications (RTCSA)

Yao S, Li J, Liu D, et al (2020b) 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)

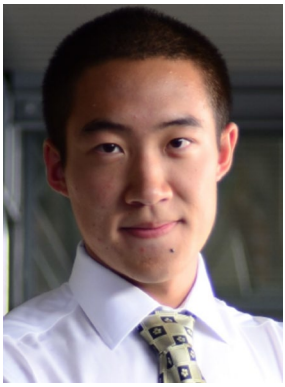
Zhang S, Lin W, Lu P, et al (2017) Kill two birds with one stone: boosting both object detection accuracy and speed with adaptive patch-of-interest composition. In: 2017 IEEE international conference on multimedia & expo workshops (ICMEW), IEEE, pp 447–452

Zhou Y, Moosavi-Dezfooli SM, Cheung NM, et al (2018) Adaptive quantization for deep neural network. In: Thirty-Second AAAI conference on artificial intelligence

Zhu X, Wang Y, Dai J, et al (2017a) Flow-guided feature aggregation for video object detection. In: Proceedings of the IEEE international conference on computer vision, pp 408–417

Zhu X, Xiong Y, Dai J, et al (2017b) Deep feature flow for video recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2349–2358

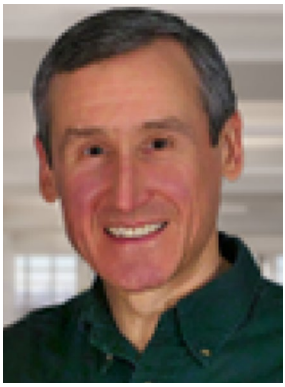
Shengzhong Liu is a postdoc research associate at the University of Illinois at Urbana-Champaign (UIUC). He received his Ph.D. from UIUC in 2021. His current research interests include machine learning for the Internet of Things (IoT) and Cyber-Physical Systems (CPS), intelligent real-time systems, deep sensor fusion, and social network analysis.



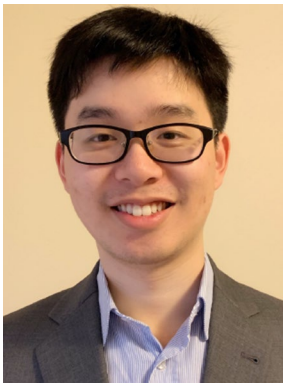
Xinzhe Fu received the B.S. degree in Computer Science from Shanghai Jiao Tong University. He is currently a Ph.D. candidate in Laboratory for Information and Decision Systems, and the Interdisciplinary Doctoral Program of Statistics at MIT. His research focuses on scheduling and optimization problems in stochastic networks.

Yigong Hu received the BS degree from Shanghai Jiao Tong University and the MS degree from Columbia University, in 2018 and 2020, respectively. He is currently working toward a PhD degree in computer science at the University of Illinois at Urbana-Champaign (UIUC). His current research interests include intelligent Real-Time Systems, Internet of Things (IoT), and Cyber-Physical Systems (CPS).

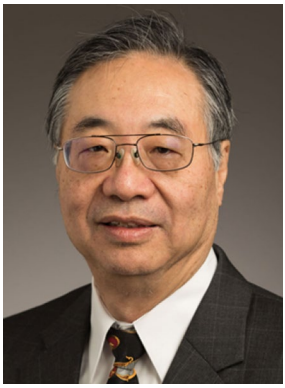
Maggie Wigness is a Senior Computer Scientist at the U.S. Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory (ARL). She earned her Ph.D. in Computer Science from Colorado State University in 2015. Maggie has led and shaped research directions in many ARL collaborative alliances including the Robotics Collaborative Technology Alliance, the Scalable, Adaptive, and Resilient Autonomy Collaborative Research Alliance (CRA), and most recently as the Collaborative Alliance Manager for the Internet of Battlefield Things CRA. Maggie's research efforts are in the cross-section of machine learning, computer vision, edge computation, and robot autonomy.



Philip David received the B.S. (1985) and Ph.D. (2006) in computer science from the University of Maryland, College Park. Since 1985, he has worked as a scientist in the Computational and Information Sciences Directorate of the U.S. Army Research Laboratory. His research is focused on providing visual perception and intelligent planning capabilities to small mobile robots. Dr. David is the author of several peer-reviewed publications spanning the areas of 3D object recognition, GPS-denied localization, machine learning, and vision and LIDAR-based perception for unmanned ground vehicles.



Shuochao Yao is an assistant professor in the Department of Computer Science at George Mason University. Yao received a PhD in computer science from the University of Illinois at Urbana-Champaign. His research focuses on building efficient and reliable artificial intelligence systems for intelligent Internet of Things (IoT) and Cyber-Physical Systems (CPS).



Lui Sha graduated with Ph.D. from CMU in 1985. He worked at the Software Engineering Institute from 1986 to1998. He joined UIUC in 1998 as a full professor. Currently, he is Donald B. Gillies Chair Professor of Computer Science Department and Daniel C. Drucker Eminent Faculty at UIUC's College of Engineering. He is a fellow of IEEE and ACM. He was a member of National Academic of Science's Committee on Certifiably Dependable Software Systems and a member of NASA Advisory Council. He led the research, development, and the transition to practice on real-time and embedded computing technologies, which were cited as a major accomplishment in the selected accomplishment section of the 1992 National Academy of Science's report, "A Broader Agenda for Computer Science and Engineering" (P.193). He led a comprehensive revision of IEEE standards on real-time computing, which have since become the best practice in real-time computing systems. Now it has been widely used in real-time systems such as airplanes, robots, cars, ships, trains, medical devices. His work on real-time and safety-critical system

integration has impacted many high technology programs, including GPS, Space Station, and Mars Pathfinder.

Tarek Abdelzaher (Ph.D., UMich, 1999) is a Sohaib and Sara Abbasi Professor of CS and Willett Faculty Scholar (UIUC), with over 300 refereed publications in Real-time Computing, Distributed Systems, Sensor Networks, and IoT. He was Editor-in-Chief of J. Real-Time Systems for 20 years, an AE of IEEE TMC, IEEE TPDS, ACM ToSN, ACM TIoT, and ACM ToIT, among others, and chair of multiple top conferences in his field. Abdelzaher received the IEEE Outstanding Technical Achievement and Leadership Award in Real-time Systems (2012), a Xerox Research Award (2011), and several best paper awards. He is a fellow of IEEE and ACM.

Authors and Affiliations

Shengzhong Liu[1] · Xinzhe Fu[2] · Yigong Hu[1] · Maggie Wigness[3] · Philip David[3] · Shuochao Yao[4] · Lui Sha[1] · Tarek Abdelzaher[1]

✉ Tarek Abdelzaher
zaher@illinois.edu

Shengzhong Liu
sl29@illinois.edu

Xinzhe Fu
xinzhe@mit.edu

Yigong Hu
yigongh2@illinois.edu

Maggie Wigness
maggie.b.wigness.civ@army.mil

Philip David
philip.j.david4.civ@army.mil

Shuochao Yao
shuochao@gmu.edu

Lui Sha
lrs@illinois.edu

1 University of Illinois at Urbana-Champaign, Champaign, USA

2 Massachusetts Institute of Technology, Cambridge, USA

3 U.S. DEVCOM Army Research Laboratory, Adelphi, USA

4 George Mason University, Fairfax, USA