Real-Time Systems
https://doi.org/10.1007/s11241-022-09387-6

Real-time task scheduling with image resizing for criticality-based machine perception

Yigong Hu[1] · Shengzhong Liu[1] · Tarek Abdelzaher[1] · Maggie Wigness[2] · Philip David[2]

Accepted: 23 June 2022

Abstract

This paper extends a previous conference publication that proposed a real-time task scheduling framework for *criticality-based machine perception*, leveraging image resizing as the tool to control the accuracy and execution time trade-off. Criticality-based machine perception reduces the computing demand of on-board AI-based machine inference pipelines (that run on embedded hardware) in applications such as autonomous drones and cars. By segmenting inputs, such as individual video frames, into smaller parts and allowing the downstream AI-based perception module to process some segments ahead of (or at a higher quality than) others, limited machine resources are spent more judiciously on more important parts of the input (e.g., on foreground objects in lieu of backgrounds). In recent work, we explored the use of image resizing as a way to offer a middle ground between full-resolution processing and dropping, thus allowing more flexibility in handling less important parts of the input. In this journal extension, we make the following contributions: (i) We relax a limiting assumption of our prior work; namely, the need for a "perfect sensor" to identify which parts of the image are more critical. Instead, we investigate the use of real LiDAR measurements for quick-and-dirty image segmentation ahead of AI-based processing. (ii) We explore another dimension of freedom in the scheduler: namely, merging several nearby objects into a consolidated segment for downstream processing. We formulate the scheduling problem as an optimal resize-merge problem and design a solution for it. Experiments on an AI-powered embedded platform with a real-world driving dataset demonstrate the practicality and effectiveness of our proposed framework.

Keywords Machine perception · Real-time scheduling · Cyber-physical systems

✉ Yigong Hu
yigongh2@illinois.edu

Extended author information available on the last page of the article

Published online: 08 August 2022

1 Introduction

This paper describes the first practical framework for sub-frame resizing as a means to implement criticality-based machine perception. In recent work (Liu et al. 2020), the authors proposed the idea that data processing in AI-based perception pipelines on edge devices, such as embedded GPUs (e.g., on drones and autonomous cars) needs to be *prioritized at a sub-frame level*, because some surrounding objects are more important than others. We call such a general framework, *criticality-based machine perception*. Processing an entire frame at a time is arguably not responsive to the relative importance of parts of the scene. For example, in a real-time context, one may argue that nearby obstacles should be inspected (by AI-based perception modules) ahead of background objects when hardware parallelism is not sufficient for doing both concurrently. Our subsequent conference paper (Hu et al. 2021) explored the use of image resizing (specifically, downsizing) at a sub-frame level as a tool for finer-grained manipulation of accuracy and execution time trade-offs in criticality-based machine perception pipelines. Resizing (to a smaller size) allows the use of *lower complexity* inference models downstream, commensurately with the reduced input object size, thereby associating different parts of a scene with different "levels of service". The added flexibility (of choosing an appropriate re-sizing level for different frame segments) was shown to give rise to real-time resource allocation algorithms that improve both responsiveness and perception accuracy for high-criticality objects (Hu et al. 2021). Compared with imprecise computation methods that use partially executed (i.e., early exit) neural networks (Yao et al. 2020), image resizing was shown to achieve better performance trade-offs thanks to more specialized neural network models and better resource utilization with improved image batching on the GPU.

Despite the aforementioned promising results, our prior conference work on the exploration of image resizing in criticality-based machine perception pipelines made a limiting assumption (Hu et al. 2021): it assumed that the input to the resizing and prioritization algorithm includes perfect bounding boxes for each object in the frame. The assumption simplified the downstream algorithms. In practice, this assumption suffers from a "chicken and egg" problem. On one hand, the resizing and prioritization algorithm must run ahead of the AI-based perception module in order to reduce the size of inputs supplied to the latter (or else no resource savings occur). On the other hand, the resizing and prioritization algorithm needs object bounding boxes not produced until the AI-based perception module is executed. In prior work (Hu et al. 2021; Liu et al. 2020, 2021), we assumed that another sensor (e.g., a ranging sensor) is used to resolve this problem; foreground objects are separated from backgrounds based on distance using a quick algorithm that does not rely on the AI-based (computationally demanding) perception module. The distance-based segments are then resized and prioritized for processing by downstream AI.

This paper extends our previous image resizing framework (Hu et al. 2021) by offering the *first practical implementation of a criticality-based perception*

pipeline that uses input resizing at a subframe level. We remove the assumption of perfect bounding boxes at the input of the resizing module. Instead, after obtaining a frame from the camera, we slice the incoming scene into regions with the help of a quick-and-dirty fast segmentation method that relies on a second sensor; a LiDAR point cloud (Bogoslavskyi and Stachniss 2017). Ordinarily, LiDAR-based point-cloud processing could be even slower than image processing. In this work, however, we do not perform actual object identification from the LiDAR point cloud. Rather, we only use LiDAR distance measurements merely to perform quick-and-dirty distance-based point clustering. Clusters of nearby points of similar distance are then mapped to segments of the corresponding camera frame and these (distance-based) frame segments are forwarded to the image resizing and prioritization module. This module determines which segments to process by subsequent AI-based perception and at what resolution.

While the above change appears to be cosmetic in nature, it leads to a fundamental rethinking of the core contribution of our previous work (Hu et al. 2021); namely, it calls for a new resizing and prioritization algorithm. The new algorithm must perform optimal (distance-based) segment consolidation together with resizing because there is no longer the assumption of one-to-one mapping between input regions (i.e., frame segments) and objects in the frame. A cluster of points that are roughly the same distance away can include multiple objects. Similarly, a large object can give rise to LiDAR points that are different distances away (over-segmentation). To adapt to the imperfect mapping of distance-based segments to objects, two techniques are applied. First, small segments are merged into larger segments that they connect to, so that over-segmented objects are restored. Second, detected bounding boxes are filtered both at the per-segment and whole-frame scale. We also observe that many segments are located in close proximity to each other in the camera frame, so it is possible to combine them together and detect multiple objects at once (using a neural network such as YOLO). Thus, we allow the scheduler to both resize and merge segments before invoking the neural network for detection. We formulate a scheduling optimization problem and design an algorithm to approximate the solution. We then look at cases where the LiDAR segmentation fails, and discuss the inherent shortcomings of using LiDAR for quick-and-dirty distance-based input segmentation. Comparing the results with our initial conference paper (Hu et al. 2021) (when fed the imperfect LiDAR inputs), the new proposed framework achieves significant improvements in execution efficiency and robustness, thanks to adapting to noisy segmentation results and considering object merging.

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 discusses the architecture of the system and the reasons for the proposed design choices. Section 4 describes the implementation details and presents the evaluation and comparison with experiments. Section 5 discusses limitations of curing with a LiDAR sensor. Concluding remarks are made in Sect. 6.

2 Related work

This work is motivated by the rise of "edge AI," where machine intelligence algorithms (e.g., deep neural networks) are introduced into Cyber-Physical Systems (CPS) and Internet of Things (IoT) applications, promoting the need for efficient execution on embedded edge devices (Yao et al. 2018a). The ability to run edge AI algorithms on embedded devices gives rise to many new applications such as self-driving vehicles (Sun et al. 2020), delivery robots (Bamburry 2015), autonomous drones (Floreano and Wood 2015), and military defense systems (Abdelzaher et al. 2018). These applications involve perception tasks such as object detection, identification, and tracking using machine learning models. Perception happens to be one of the more resource-consuming machine intelligence tasks. It is no coincidence that after thousands of years of biological evolution, more than 50% of the human brain cortex (the part of the brain typically associated with higher-level cognitive functions) is the visual cortex (i.e., a perception subsystem). The resource demands of machine perception models make them a major bottleneck to system performance (Alcon et al. 2020; Lin et al. 2018).

Recent work has focused on reducing the neural network model size and increasing perception speed. Examples include parameter pruning techniques to remove small-weight connections (Han et al. 2016); pruning network structure with a compressor-critic framework (Yao et al. 2017); incremental learning based on a grow-and-prune neural network synthesis paradigm (Dai et al. 2020); quantizing weights and activations to adapt to different resource budgets (Jin et al. 2020); speeding up convolutional layers with approximations using linear structure present within the convolutional filters (Denton et al. 2014); leveraging the sparsification of fully connected layers and separation of convolutional kernels to reduce the resource requirements (Bhattacharya et al. 2016); projecting the filter channels to a unified low dimensional space (Minnehan et al. 2019); and compressing convolutional neural networks in the frequency domain (Wang et al. 2016). Combining several of the compression and acceleration methods can potentially achieve superior performance (Han et al. 2016; Jung et al. 2019).

While the above solutions have been effective at reducing neural network resource demands, they are generally not concerned with real-time execution under criticality and/or time constraints. To address this gap, work on real-time systems offered optimizations and approximations of deep neural networks with consideration for real-time performance. For example, Yang et al. (2019) reimplement the YOLO neural network to optimize system utilization and GPU workload allocation on NVIDIA DRIVE PX2 to achieve higher throughput; Bateni and Liu (2018) propose an approximation approach that trades off accuracy and cost on a per-layer basis to guarantee deadlines of DNN workloads; and Heo et al. (2020) build a worst-case execution time model for DNNs on a GPU and propose a multipath network that can dynamically select different execution paths at runtime to meet time constraints. Other flavors for adaptively trading off computation time and solution quality have been proposed (Kim et al. 2020a), including a framework for learning abstract information early and learning more concrete

information as time allows (Kim et al. 2020b) and a framework for casting neural network workflows as imprecise computations (Yao et al. 2020).

Efforts have also been made to understand the GPU platform and firmware behavior, so as to optimize system performance. For example, Otterness et al. (2017) evaluate the suitability of the NVIDIA TX1 platform for real-time computer vision workloads and (later) suggest AMD GPUs as a viable alternative for real-time GPU research (Otterness and Anderson 2020); Yang et al. (2018) identify the pitfalls of the NVIDIA CUDA GPU software and provide best practices for applying real-time safety-critical principles; Olmedo et al. (2020) investigate hierarchical scheduling policies of NVIDIA GPUs and their proprietary CUDA application programming interfaces; and Yao et al. (2018b) identify non-linear relations between neural network structure and execution time, and exploit them to find network configurations with the best trade-off between execution time and accuracy.

The above solutions adapt neural network structure to better optimize for the urgency and criticality of inputs. However, they do not consider *segmenting the input at a sub-frame level* in a manner that separates segments of different criticality, such that different levels of adaptation could be applied to different segments. Liu et al. (2020) propose a prioritization pipeline to remove priority-inversion in machine perception pipelines by slicing the input frame into sub-areas of different priorities. Our conference publication (Hu et al. 2021) then investigates criticality-based input resizing as a universally applicable adaptation to enhance the flexibility of trade-offs between perception quality and timeliness.

Criticality-based machine perception relies on cues (e.g., from another sensor) that allow quick identification of regions of interest in each frame (Liu et al. 2020). However, these cues must themselves be computed very efficiently, calling for “quick-and-dirty" algorithms. As a result, the accuracy of these cues is (in practice) relatively low; a topic that has not been the focus of prior our work. We extend our conference publication (Hu et al. 2021) by relaxing the assumption of perfect data segmentation, and changing the scheduling algorithm accordingly.

Work that bears some (superficial) similarity to this paper is a recent arXiv manuscript on LiDAR-based segmentation and resizing (Chen et al. 2021). However, the resizing approach in that work is *not criticality-based*. Instead, it is purely quality-based. Closer objects are down-sized more, irrespective of their criticality, as opposed to downsizing less important parts of the frame. We also investigate the general limitations of cueing with a LiDAR sensor.

3 System architecture

In this section, we describe the system architecture and the reasons behind our design choices. We consider an application running on an embedded platform equipped with a camera and a LiDAR sensor. The application requires timely processing of the input frames to detect and classify objects in the surrounding environment. A data slicing module provides noisy bounding boxes generated from the LiDAR measurements and a scheduling algorithm makes optimal resizing and merging decisions. An overview of the system architecture is shown in Fig. 1.

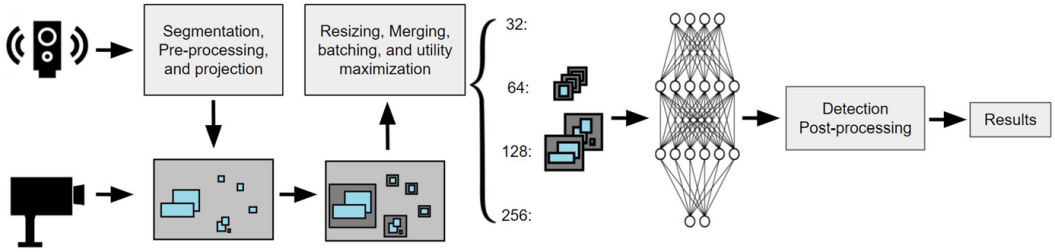


Fig. 1 An overview of the system architecture

3.1 LiDAR segmentation and pre-possessing

We apply real-time LiDAR point-cloud-based object segmentation methods (Bogoslavskyi and Stachniss 2017) to slice the input frame and generate candidate bounding boxes. Due to the physical properties of the laser beams used in LiDAR sensors, the measurement results can be easily affected by factors including the multi-path effect, the object surface type, and the weather. Also, compared with image sensors, LiDARs usually have a much shorter operation range and lower resolution. In order to achieve a very small computation overhead, the segmentation module has to run in real-time on the CPU, preventing the use of more sophisticated neural network-based LiDAR semantic segmentation methods (Milioto et al. 2019). As a result, the segmentation algorithm is unable to distinguish between objects of interest such as cars, from other objects such as trees. Moreover, the output could be over-segmented, i.e., one object could be split into several segments, or under-segmented, i.e., several objects may appear in one segment.

Since multiple objects may appear in one segment, simply using a classification model to inspect each segment is no longer adequate. Instead, a detection model such as YOLO is needed because of its ability to detect multiple objects in a scene. Using YOLO as the inference model could filter out non-target objects. However, accurate detection of an object requires a segment to cover the entire region of the object. YOLO is capable of detecting multiple objects within one segment, but over-segmented objects need to be merged first. In addition to affecting detection accuracy, over-segmenting produces a larger number of segments, increasing the workload of the inference model. Thus, pre-processing the LiDAR segmentation is necessary. We start from the largest segment and merge any smaller neighboring segments that are within a 3D distance threshold to it. Any segments that are smaller than a predefined size threshold and are not merged into a larger segment are removed, as they are most likely not an object of interest. We observe that when the bounding boxes are fit tightly to the objects, YOLO does not produce the best detection results, likely because YOLO is trained to detect smaller objects from larger images. Thus, we expand each segment slightly for better detection performance. An example of a frame region before and after pre-processing is shown in Fig. 2.

Fig. 2 An example of a segment before (**a**) and after (**b**) pre-processing

3.2 Segment merging

Bounding boxes of segments in the camera frame may overlap with each other. Instead of processing each of the overlapping segments separately, we can merge several closely located segments into a consolidated region and process them together. As a result, fewer segments are needed to cover all objects present in the camera frame. However, if the size of a merged box increases into a larger quantized size, the increase in processing time will be greater than the combined execution time of several separated smaller objects and merging is not beneficial. Furthermore, a merged box will be downsized if it exceeds the largest quantized size, and the detection accuracy will be affected. Considering these observations, we only allow merging if the merged segment is smaller than the largest allowed size and still within the same quantization interval. We start from the largest segment and iteratively find and merge segments within its proximity. Once a segment is merged into another segment, it is not eligible to be merged again so that one object will not be detected multiple times. Segments in a merged region will be detected together if they all have the same resize scale and detected separately otherwise. We call these rules the merging constraints. An example is shown in Fig. 3, where the green bounding boxes are merged together into the red bounding box. The largest segment is quantized to 256×256 size and the merged bounding box is also 256×256, thus satisfying the merging constraint. When all objects have the same resize scale, all the segments within the red bounding box are processed together. In contrast to batching where several segments of the same size are processed in parallel, merged segments only require running YOLO once to detect multiple objects.

3.3 Image resizing

Our previous work (Hu et al. 2021) argues that imprecise computation based approaches (Yao et al. 2020) (that vary the number of deep neural network stages to adapt quality depending on input) are inferior to model switching approaches that

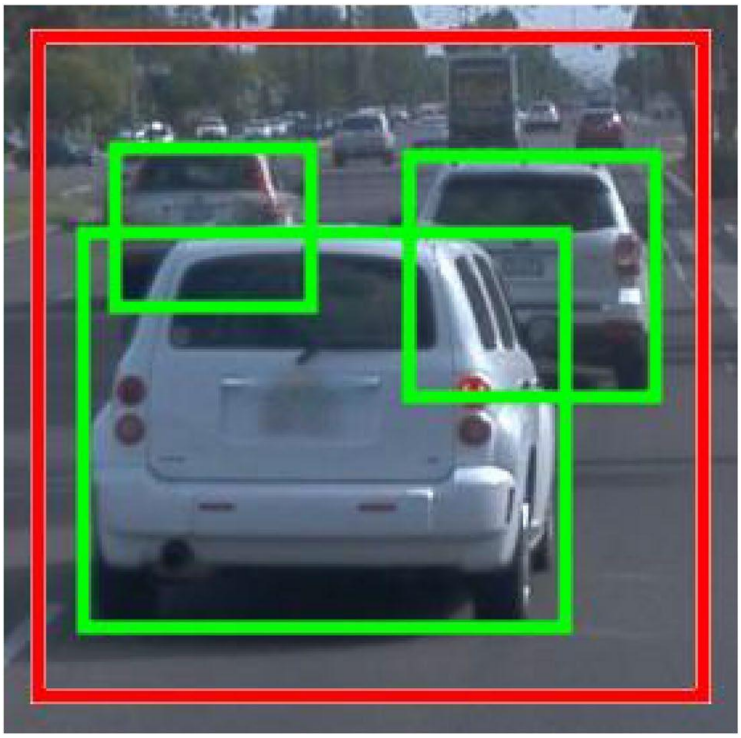


Fig. 3 An example of bounding box merging

simply pick a different neural network model from a range of alternatives to offer different points in the quality/latency trade-off space. In addition to that, imprecise computation does not extend well to some architectures. For example, YOLO introduces different key functions at different downstream parts of the network, making the separation of mandatory and optional components quite complicated. In our design, we choose to use YOLO as the perception model in order to accommodate noisy segments generated with LiDAR measurements, and use image resizing as a tool to control the modulating of execution time assigned to input segments of different criticality. The average accuracy when resizing objects of different sizes is profiled offline as an input to the scheduling algorithm.

3.4 Batching

The GPU of the embedded platform is capable of executing multiple jobs in parallel, by processing inputs (e.g., multiple segments) in batches. On some lower-end *embedded* device GPUs, batching is often constrained by the requirement that all inputs must be processed by the same processing kernels [34]. Since the choice of kernel depends on the input size, in practice this means that (processing of) only same-size images can be batched together. We observe that when the number of images batched together increases, the total inference time increases but at a slower rate than that for sequential execution (until the GPU gets fully utilized). Thus, it is always beneficial to batch as many images (of the same size) as possible together.

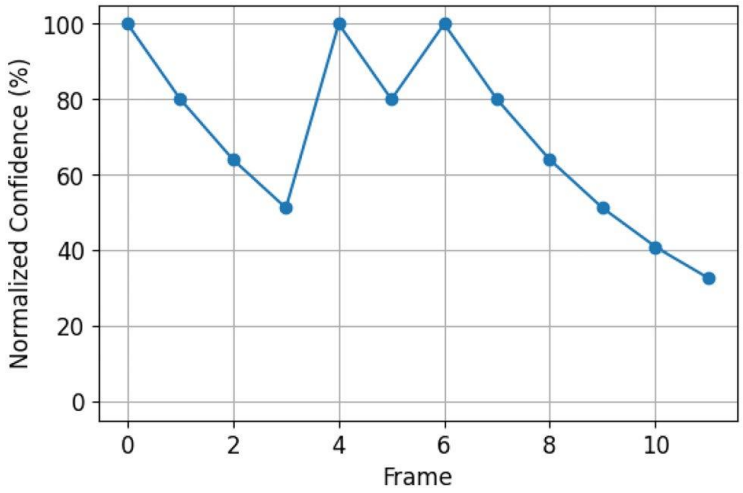


Fig. 4 The change of classification confidence with tracking

In other words, consider a collection of n images with the same size s. Let the total time required to process this batch be denoted by $T(s, n)$. If we divide the batch into two sequential batches, each containing n_1 and n_2 images, respectively, where $n_1 + n_2 = n$, then the time needed to execute these two batches sequentially is $T(s, n_1) + T(s, n_2) > T(s, n)$. This observation indicates that maximum batching should always be used for same-size inputs. This insight reduces the problem complexity.

3.5 Redundancy between frames

When the sampling rate of the sensors is relatively high, there will be a large amount of redundancy between frames. An object that appeared in the previous frame will likely also appear in the current frame. Thus, opportunities arise to *track* an object's trajectory given its identified class and location in a number of previous (not necessarily consecutive) frames. In turn, such a trajectory can be extrapolated into the future to yield a region, where the object might currently be found. When a new frame arrives and a sensor, such as a LiDAR, identifies all regions of interest (e.g., parts of the point cloud that are at a different distance from their surroundings, suggesting the presence of an obstacle), if one such region coincides with the expected location of a previously tracked object, one may assume that the object has been (re-)identified and skip sending the segment to the perception module. There is presently a large volume of literature on object tracking that allows computing future expected locations of objects (Bar-Shalom and Tse 1975). A review of these solutions is outside the scope of this paper. We simply use their results to compute tracking confidence and consequently determine whether or not a given segment needs to be sent to the perception module. This determination is part of the optimization problem formulated in Sect. 3.7, where the scheduling model is discussed.

If a segment is inspected by the perception module, the object class is determined with some (classification) confidence. If the segment is not sent (on account of it being found a continuation of a previous trajectory), the previous classification

Table 1 Table of notations

Symbol	Meaning
H	Frame interval
$\mathcal{T}$	Available jobs
τ_i	Sub-task related to the i-th object
i	Object index
$\mathcal{S}$	Set of legal image sizes
s	Image size
n	Number of images in a batch
k	Number of frame
$C_{i,k}$	System classification confidence of object i in frame k
E_{s_i}	Model classification confidence of object i with size s
$F_{i,k}$	Tracking confidence of object i in frame k
s_i^0	Original size of the i-th object
w_i	Criticality weight for object i
$R_{i,s}$	Utility of executing object i with size s
U	Total utility
$G_{i,s}$	Incremental efficiency at size s of task i
$T_{s,n}$	Total time to process a batch of n images with size s
B_s	Slope of linear fit for total processing time
D_s	Intercept of linear fit for total processing time
$x_{i,s}$	Schedule choice for the i-th object and new size s

confidence for that object is discounted by the additional uncertainty arising from the possibility of misclassification due to association errors in tracking (as would be common in multi-target tracking environments).

Figure 4 shows an example of the confidence trail in the classification of an object. At times 0, 4, and 6, the object is processed by the perception module, and the inference results are updated. At other times, the perception task is skipped and the confidence decays from frame to frame. Some computation is thus saved by leveraging tracking, at the cost of potential classification error caused by incorrect object association across frames. A mathematical formulation of the underlying optimization is discussed later in Sect. 3.8.

3.6 Criticality weight and utility

Many heuristics are possible for the assignment of criticality weights to segments. This topic is orthogonal to our work in that no matter how one decides to assign criticality, we offer a mechanism for allowing higher criticality segments to receive preferential treatment. For better generalizability, we incorporate an abstract criticality weight into our utility function and let the scheduling algorithm optimize the cumulative utility of the executed jobs. In our implementation, we use (a function of) object distance from the sensor as the criticality value. The scheduling problem is now described more formally below (Table 1).

3.7 Scheduling problem

Consider a perception system that processes segments of successive frames on a GPU. Let H denote the frame interarrival interval (or *frame interval* for short). The processing of one segment is referred to as one *perception task*. In our implementation, the perception task executes the YOLO neural network on one segment to generate an object detection result. The main assumption we require is that the neural network architecture must accommodate inputs of different sizes. YOLO is a fully convolutional network and allows for multiple sizes of inputs (selected from a predefined set of allowable sizes). Each input size corresponds to a different network structure, where smaller sizes result in smaller neural networks. As is common to lower-end GPUs, only same-size inputs can be batched (i.e., executed concurrently). This is because lower-end GPUs generally require that the same kernel be executed on all threads. Since frames arrive periodically, all segments of one frame must either be processed or discarded before the next frame arrives.

Consider a single scheduling period corresponding to the processing of one input frame, k. We call it period k. We denote the set of arrived jobs (input segments generated by LiDAR processing) within a scheduling period k by set $\mathcal{T}[k]$. A single task is denoted by $\tau_i[k] \in \mathcal{T}[k]$. The function of the task is to process one segment of the frame by the perception module. Let $\mathcal{S}$ denote the set of legal image/segment sizes (that the neural network of the perception module can accept). All segments must first be padded to fit one of the legal sizes (which is done in a best-fit manner). In order to improve batching, only a handful of different sizes are supported. Let the segment corresponding to task $\tau_i[k]$ have an initial (padded) size $s_i^0[k]$ and a criticality weight $w_i[k]$. The criticality weight might be assigned, for example, based on the distance to the object that appears in that segment, as computed by the LiDAR. The scheduler is allowed to resize the input image to a size $s_i[k] \in \mathcal{S}$, that satisfies, $s_i[k] \leq s_i^0[k]$, in order to save processing time, at the cost of lowering accuracy. Also, the scheduler is allowed to merge objects together if they satisfy the merging constraints, where close-by objects with the same resize scale are combined for detection. The merging decision is denoted as $\mathcal{M} = \{M_1, M_2, ..., M_j\}$, where M_j is the set of segments to be merged together. When segments are merged, only one run of YOLO on the merged segment is required to detect all objects in the segment.

Let E_{s_i} be the average model classification confidence for an input segment of size s_i (based on prior empirical testing of the neural network).[1] Also, let $F_{i,k}$ be the multiplicative drop in tracking confidence for object i, from frame to frame, in the absence of re-identification by the perception module. For each segment that matches a previous object trajectory, the scheduler can either choose to execute the perception task (possibly after resizing) or use the previous object classification result (after discounting it by $F_{i,k}$). In the latter case (i.e., if the task is not selected for execution), for notational convenience, we say that the selected

[1] Note that in the presence of multiple objects in the segment, this empirically computed number can be the average of per-object confidence values.

resizing is $s_i = NULL$. The estimated confidence, at interval k, in the classification of the object in segment, i, denoted $C_{i,k}$, is thus computed as follows:

$$C_{i,k} = \begin{cases} F_{i,k} C_{i,k-1} & \text{if } s_i = NULL \\ E_{s_i} & \text{otherwise} \end{cases} \tag{1}$$

The utility collected by a task operating on input of size, s_i, denoted $R_{i,s_i}[k]$, is then defined as:

$$R_{i,s_i}[k] = w_i C_{i,k} \tag{2}$$

where w_i is the criticality weight (higher for more critical objects), and $C_{i,k}$ is the confidence, computed from Equation (1). Note that, the utility expressed in Equation (2) depends on three scheduling choices for each task:

- Whether to run the task or not (i.e., whether $s_i = NULL$ or not). This choice determines the estimated $C_{i,k}$ that enters the utility calculation.
- How the input is resized (what value to choose if s_i is not $NULL$). This choice affects E_{i,s_i} and thus $C_{i,k}$.
- The set of objects to be merged and detected together. This merging decision $\mathcal{M}$ affects the execution time of the chosen segments.

The algorithm optimizes the total utility of all jobs in frame k. We call it the *optimal resizing-merging problem*, defined below.

The Optimal Resizing-Merging Problem: *In each scheduling period, k, of length H, given a set of arrived perception jobs, $\mathcal{T}[k]$, where each task $\tau_i[k] \in \mathcal{T}[k]$ has a criticality weight, w_i, and a classification confidence $C_{i,k}$, given by Equation (1), find an optimal (resized) input size s_i (no bigger than the original) for each $\tau_i[k]$ and segment merging choices $\mathcal{M}$, such that all jobs are processed within interval H, and the sum of utilities is maximized. In other words, find:*

$$\arg\max_{\forall i: s_i} \sum_{\tau_i[k] \in \mathcal{T}[k]} R_{i,s_i}[k] = w_i C_{i,k}$$

3.8 Solution algorithm

The total time required to process a batch of n images of size s is denoted by $T(s, n)$. This value is available from offline system profiling for the worst-case execution time. Let $x(i, s) \in \{0, 1\}$ indicate whether or not the i-th task is executed with input resized to size s. Skipping the execution of task, i, is denoted by setting $s = NULL$ (i.e., $x(i, NULL) = 1$). Thus, for any one task, $\sum_s x(i, s) = 1$, since the corresponding segment is either omitted altogether ($s = NULL$), or resized to exactly one size. For each image size s, the batch size n_s is calculated considering the merging choices. The mathematical formulation of the optimization problem becomes:

$$\max_{x} \sum_{i} \sum_{s} x(i,s) R_{i,s} \tag{3}$$

$$\mathbf{s.t.}\ \forall i, \forall s > s_i^0 \ : \ x(i,s) = 0, \tag{4}$$

$$\forall i, \forall s \ : \ x(i,s) \in \{0,1\}, \ \sum_{s} x(i,s) = 1, \tag{5}$$

$$\sum_{s} T(s, n_s) \leq H, \tag{6}$$

where:

$$\forall s \ : \ n_s = \sum_{\tau_i \notin \mathcal{M}} x(i,s) + \sum_{M \in \mathcal{M}} x(\arg\max_i s_i^0, s). \tag{7}$$

In the above formulation, (3) is the optimization objective, (4) constrains that input segments can only be downsized; (5) constrains that each segment can only be processed at one size (or dropped); (6) constrains that the total time cannot exceed the frame interarrival interval, H; and (7) simply defines the number of images sized s, considering the merging decisions $\mathcal{M}$.

We apply reasonable approximation to simplify the scheduling problem for real-time execution. We observe that the total executing time for a batch $T(s, n_s)$ of images increases close to linearly with the number of images in the batch. We fit a linear function to the total execution time, resulting in the empirical approximation, $\hat{T}(s, n_s)$, given by the expression:

$$\hat{T}(s, n_s) = D_s + B_s n_s \tag{8}$$

where D_s and B_s are constants, obtained from linear regression, that depend only on image size, s. Note that the linear increase of $\hat{T}(s, n_s)$ with batch size, n_s, is not attributed to sequential execution (since the batch is processed in parallel). Rather, the linear slope is attributed to copying data into and out of the GPU. Naturally, the amount of data copied grows linearly with the size of the batch. Adding up Eq. (8) overall legal image sizes, we get:

$$\sum_{s} \hat{T}(s, n_s) = \sum_{s} D_s + \sum_{s} B_s n_s \leq H \tag{9}$$

where the inequality (on the right) is from Eq. (6). The inequality can be rewritten as:

$$\sum_{s} B_s n_s \leq H - \sum_{s} D_s \tag{10}$$

Substituting for n_s from Eq. (7), we get:

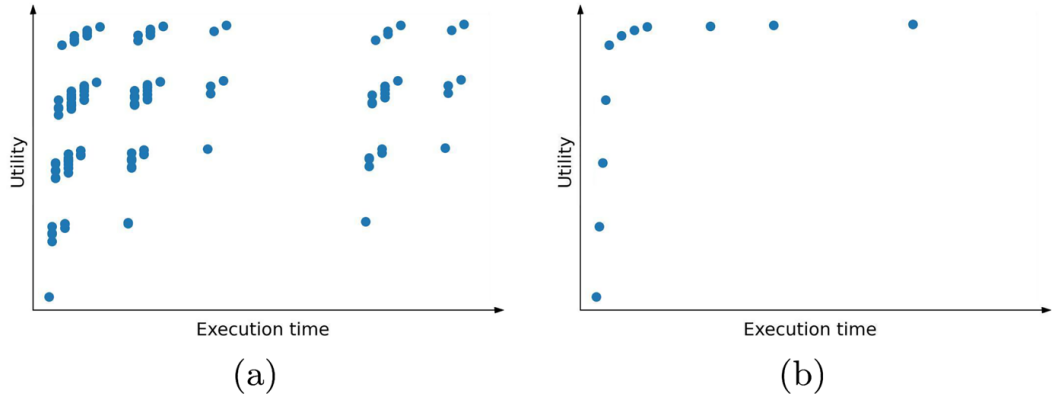


Fig. 5 Resize combinations before (**a**) and after (**b**) LP-dominance removal

$$\sum_{s}\sum_{i} x(i,s)B_s \leq H - \sum_{s} D_s \tag{11}$$

Let us call the right-hand-side, $\hat{H} = H - \sum_s D_s$. Note that the right-hand side is a constant. The original optimization objective (3), combined with the above constraint forms the optimization problem. Due to added dimension of freedom for merging, solving the optimization problem is hard. In order to simplify the problem, for each available merging set M, we consider all the possible combinations and treat each of them as a choice for the merged segment. Each item $v \in V = \{1, 2, 3, ...\}$ in this class represents a combination of resizing and merging choices, with a corresponding time cost t_{Mv} and utility gain R_{Mv}. Then we can formulate the optimization problem as a Multi-Choice Knapsack Problem (MCKP) (Kellerer et al. 2004), where the goal of the MCKP is to choose from a set of multiple-choice items, and choices are limited by constraint (5), such that the sum of "rewards" (3) is maximized.

The MCKP problem is NP-hard, as it contains the well-known NP-hard knapsack problem as a special case. As a result, it is not possible to solve the problem in real time due to its high complexity. One solution is to approximate the problem by quantization and use dynamic programming to solve it in polynomial time. In practice, we find the overhead of this solution unacceptable for sufficiently fine-grained quantization. Instead, we adopt a greedy algorithm to approximate the optimal solution. The greedy algorithm starts with an empty set of jobs ($s = NULL$ for all), and incrementally upgrades the task with the highest incremental reward per unit of incremental cost. When we consider merging, in order to apply the greedy algorithm, LP-dominated resizing-merging choices in each merged segment need to be removed so that the remaining choices form an upper left convex hull on the utility-time plane. LP-dominance between items $j, k, l \in V$ is defined as: if items j, k, l with $t_{Mj} < t_{Mk} < t_{Ml}$ and $R_{Mj} < R_{Mk} < R_{Ml}$ satisfy: $\frac{R_{Ml}-u_{Mk}}{t_{Ml}-t_{Mk}} \geq \frac{R_{Mk}-u_{Mj}}{t_{Mk}-t_{Mj}}$,then item k is LP-dominated by items j and l. When calculating the points on the upper left convex hull, instead of calculating execution time and utility for all possible combinations, we only need to consider the best resizing choices for executing all objects separately and all the merging options. An example with execution time

and utility of combinations in a segment before and after LP-dominance removal is shown in Fig. 5.

Now, for each segment we define:

$$\hat{R}_{i,s} = \begin{cases} R_{i,s} - R_{i,s-1} & \text{if } s > s_{min} \\ R_{i,s} & \text{if } s = s_{min} \end{cases}$$

and

$$\hat{B}_s = \begin{cases} B_s - B_{s-1} & \text{if } s > s_{min} \\ B_s & \text{if } s = s_{min} \end{cases}$$

where $s-1$ denotes the size one level smaller than s. Then we calculate the incremental efficiency as $G_{i,s} = \hat{R}_{i,s}/\hat{B}_s$ for each task i and size s. We sort $G_{i,s}$ for all i, s in decreasing order, with each value of $G_{i,s}$ associated with the indices i, s during the sorting. The incremental efficiency for resizing-merging choices in a merged segment containing multiple segments is calculated in similar manners.

Next, we iteratively pick out the highest incremental efficiency $G_{i,s}$, update the corresponding size decision for task i: $x_{i,s} = 1$, $x_{i,s-1} = 0$, and calculate the amount of time left as $\hat{H} = \hat{H} - \hat{B}_s$. When a resizing-merging combination in a merged segment is chosen, the decision and execution time are also updated accordingly. The incremental efficiency $G_{i,s}$ within each task τ_i monotonically decreases as execution time increases. This ensures that for each class, a smaller execution time will always be picked first. When the time is used up, we get a near-optimal scheduling decision. The jobs with the same new sizes are batched together and executed on the GPU. Algorithm 1 presents the pseudo-code. We should observe that the above greedy solution is among the standard heuristics for solving multiple-choice knapsack problems (Sinha and Zoltners 1979).

A careful reader might have noticed a fallacy in the presented approach. Namely, in Eq. (11), the summation term on the right-hand side depends on the number of batches used. Specifically, it should range only over the values of D_s for those image sizes, s, for which image batches have been constructed after resizing. One batch is constructed per distinct size. However, we do not know which sizes have been chosen before running the optimization. In some cases, the optimal resizing might not utilize all sizes, resulting in a number of batches that are smaller than the number of supported image sizes. For example, when the scheduling period is very short, the optimal decision could be to resize every segment to the smallest size. If we still subtract D_s for the sizes not chosen, time will be wasted. With four possible sizes supported by ResNet, there are $2^4 = 16$ possible combinations of sizes used.

Solving 16 optimization problems (and comparing achieved utility to find the global maximum) is costly. Instead, we loop through a linear number of cases, where we limit the maximum image size (e.g., to 32×32, 64×64, 128×128, and 256×256 for ResNet) and assume that all sizes up to that size are used. We pick the solution with the highest cumulative utility as the optimal solution of the linear number of solutions considered.

Algorithm 1 Proposed Scheduling Algorithm

Input Available task set $\mathcal{T}$, execution time function variables D and B, utility of jobs R, frame interval H

Output Optimal task and size choice x, optimal total utility U

$\hat{H} = H - \sum_s D_s$, $U = 0$
Remove LP-dominant resizing-merging choices for merged segments
for τ_i in $\mathcal{T}$ **do**
    for s *in* $\mathcal{S}$ **do**
        Calculate $\hat{B}_{i,s}$, $\hat{R}_{i,s}$ and
        $G_{i,s}$
    end for
end for
P = Sort((i, s), order = $G_{i,s}$, inverse = True)
for p in P **do**
    $\hat{H} = \hat{H} - \hat{B}_{i,s}$
    if $\hat{H} < 0$ **then**
        break
    else
        $x(i, j) = 1$
        $x(i, j-1) = 0$
        $U = U + R_{i,s}$
    end if
end for

3.9 Detection post-processing

Because objects may overlap in the camera frame, in addition to the targeted object, a frame segment often contains incomplete parts of other objects. Also, an object can be present in multiple frame segments. As a result, multiple (sometimes partial) detections of the same object will be produced in different segments. We filter these bounding boxes in two steps. First, for each segment, we remove all the detections that intersect with the segment edges, as these detections correspond to objects splitted between segments. An example is illustrated in Fig. 6. Then we apply a standard global non-maximum suppression (NMS) step on the entire frame to remove all the remaining repeated detections.

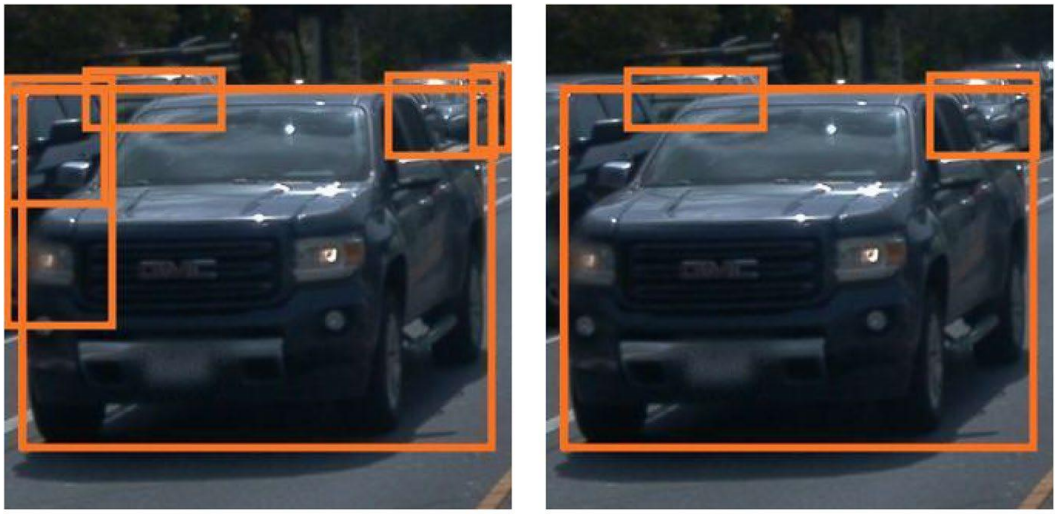


Fig. 6 The detection result of a segment before (**a**) and after (**b**) filtering

4 Evaluation

4.1 Experiment setup

4.1.1 Hardware platform

To test the above algorithm. we use an NVIDIA Jetson AGX Xavier SoC, which is an AI-powered embedded platform widely used in the industry. It is equipped with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory shared by both the CPU and the GPU. When running at 30 Watts, the Jetson AGX Xavier can deliver over 30 TOPS for deep learning applications. We set the GPU to run at a constant clock frequency for more stable performance.

4.1.2 Dataset

We perform our experiments on the Waymo Open Dataset (Sun et al. 2020), which is a large-scale autonomous driving dataset collected with self-driving cars in various conditions. Each data sequence in the dataset has a length of 20 seconds, with synchronized camera frames and LiDAR measurements collected at 10 Hz. There are 4 types of objects in the dataset: vehicles, pedestrians, cyclists, and signs. We only use the front camera data for our experiment. The objects of different sizes are rounded up and padded to 32×32, 64×64, 128×128, and 256×256, to be processed by YOLO.

4.1.3 Neural network

We use YOLOv5[2] for detecting and classifying objects in the segments. For each set of experiments, we have four networks with input sizes at 32×32, 64×64, 128×128, and 256×256, respectively, to process segments with various sizes. The models are trained with the COCO dataset (Lin et al. 2014), which contains 80 classes, and the output is mapped to the object types of the Waymo Open Dataset. To ensure fairness of comparison and to isolate the effect of design differences, we use the same model for all scheduling algorithms.

4.1.4 Load and evaluation metrics

Similar to our conference publication, we follow the method in Liu et al. (2020) and vary the frame interval from 40ms to 160ms, to simulate different loads for the scheduling algorithm. This is equivalent to replaying the recorded video segments in “slow motion" or “fast-forward" mode. We compare the average normalized accuracy, average response time, and deadline miss rate for each algorithm. We consider a task to miss its deadline if it is not executed within the frame interval, H. We define normalized accuracy as the ratio between the achieved mAP and the full-frame detection mAP with the original frame resolution.

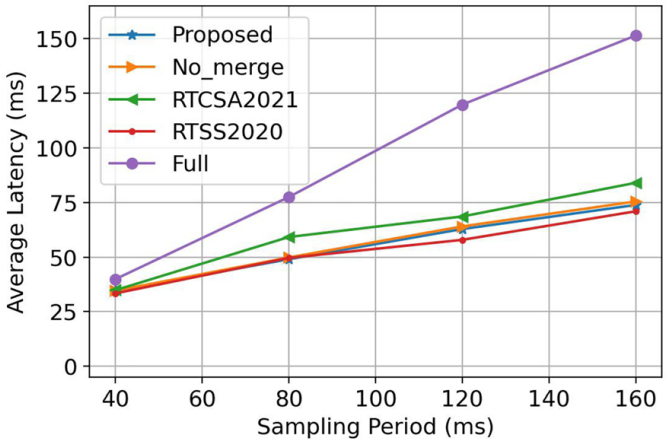
4.2 Scheduling algorithms comparison

We compare the performance of different scheduling algorithms under the same settings, with different frame intervals. The compared algorithms include:

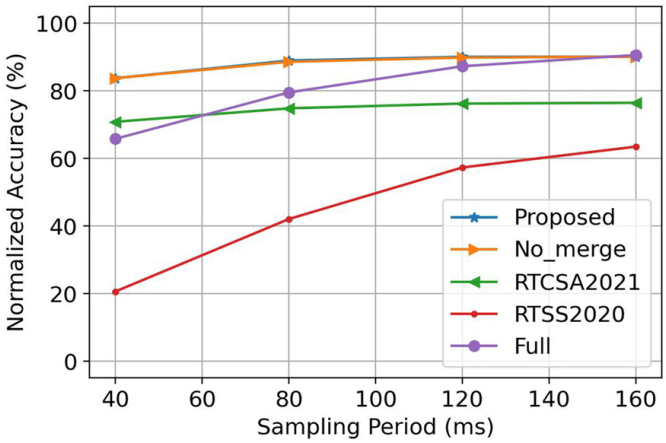
- **Proposed**: The proposed greedy scheduling algorithm with resizing and merging. It uses our greedy heuristic to choose a near-optimal subset of jobs, their new sizes, and corresponding merging options, and batches the jobs with the same new size together for execution.
- **No-merge**: The proposed algorithm with merging turned off. It still applies the empirical optimizations to adapt to noisy LiDAR segmentation results.
- **RTCSA2021**: Our previous work using a greedy scheduling algorithm with resizing. We use LiDAR-generated bounding boxes and replace ResNet with YOLO as the perception model. No empirical optimization is used.
- **RTSS2020**: The greedy scheduling algorithm proposed by Liu et al. (2020). We use LiDAR-generated bounding boxes and replace ResNet with YOLO as the perception model. Imprecise computation is not available because of the use of YOLO. No empirical optimization is used.
- **Full**: It resizes the whole camera frame to the largest size that can finish within the frame interval and use YOLO to detect the objects. It serves as a baseline for comparison.

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

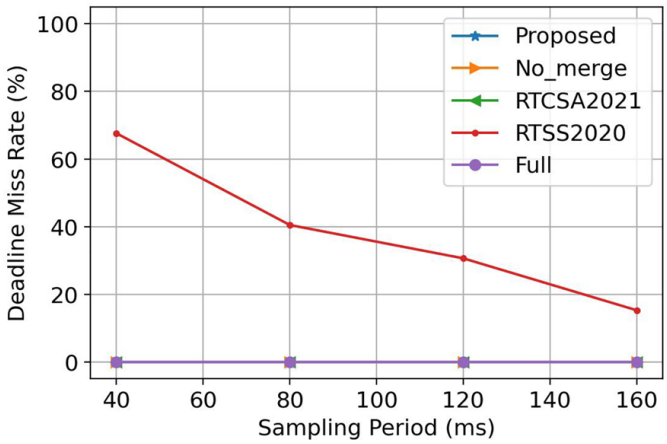
Fig. 7 Evaluation results comparison



(a) Average inference latency.



(b) Average normalized accuracy.



(c) Average deadline miss rate.

We evaluate the scheduling algorithms in terms of achieved classification accuracy, average latency, and deadline miss rate. A deadline is considered missed if a task does not execute (i.e., $s = NULL$) *and* the object has not been seen before.

This is as opposed to a situation where a task is not executed because tracking decides to inherit object classification from the previous frame. This way, we do not penalize the algorithm for skipping a task due to temporal redundancy. The results are presented in Fig. 7. Most of the time, the Proposed algorithm outperforms the other algorithms by a clear margin in terms of normalized accuracy, especially when the frame interval is shorter. Only when the sampling rate is much lower, full-frame detection could catch up because it is allowed enough time to run YOLO on the full frame on a larger scale. The difference between RTCSA2021 and No-merge is solely attributed to the empirical optimizations. Because of the lack of empirical optimization to adapt to the noisy segmentation results, RTSS2020 and RTCSA2021 cannot achieve desired accuracy even if the sampling rate is low.

All the compared algorithms except RTSS2020 achieve no deadline misses for all tested frame intervals. The algorithms with resizing ability can resize all segments to the most efficient size and execute all of them in one batch. RTSS2020, however, no longer has imprecise computation as a tool to control execution time and accuracy trade-off and is essentially reduced to a greedy algorithm that always picks the segment with the highest utility gain. This demonstrates the importance of the scheduler's ability to trade off between inference quality and execution time. On the other hand, resizing is a more versatile approach for controlling the execution time assigned to different segments. When the sampling period is very short, the scheduler is able to resize all the segments to the most efficient 32×32 size and process them all together in one batch.

As the sampling period extends, the average latency for inference of all algorithms increases. While the average latency of full-frame detection closely follows the sampling period, that of all other algorithms increases at a slower rate, as more segments are executed at larger sizes. Note that the difference between Proposed and No_merge is attributed solely to segment merging. As can be seen, segment merging improves accuracy and reduces inference latency at the same time. However, as only a small number of segments can be merged together, the improvement is not significant.

4.3 Scheduling algorithm overhead

As mentioned in Sect. 3, the greedy algorithm has a much smaller time complexity than the dynamic programming algorithm. Its time complexity is $O(n \log n)$, where n is the number of all possible choices. We evaluate the overhead of the proposed scheduling algorithm with different numbers of objects in the scene. The results in Fig. 8 show that the algorithm takes less than 12 ms for scheduling 70 objects, which accounts for a small fraction of the scheduling period. It is also worth noting that scheduling runs on the CPU, whereas machine perception runs on the GPU. Thus, perception accuracy and throughput will not be affected as long as the scheduling algorithm runs faster than the frame interval.

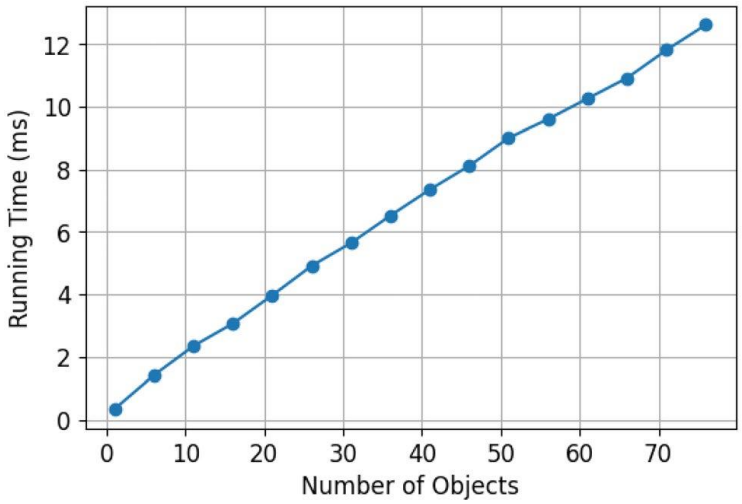


Fig. 8 Scheduling overhead

5 Limitations and discussion

Optimizing real-time perception systems running on embedded platforms is a challenging technical problem. Our proposed algorithm can significantly improve the trade-off between perception accuracy and response time. Nevertheless, there are several limitations of our approach, especially using LiDAR as the cueing sensor. First, we observe that the LiDAR sensor only covers a limited portion of the camera scene. Specifically, the LiDAR only scans the lower half of the scene closer to the ground. This is not a problem when the car is on flat ground. However, when there is an uphill slope in front of the car, the LiDAR will fail to scan the objects at a higher position. Second, the LiDAR sensor has a limited operating range. For example, the long-range LiDAR used in the Waymo Open Dataset only covers a maximum of 75 meters. The camera is able to pick up very far away objects, but objects out of the range will never be detected by the LiDAR. As a result, those objects will be omitted when LiDAR cueing is used. Lastly, the LiDAR sensor is more sensitive to the ambient environment, because of the physical properties of the laser beam. The multi-path effect, the object surface type, and the weather can easily affect the performance of the LiDAR. Because of these problems, using LiDAR as the cueing sensor will inevitably affect the system's performance. Instead, other more robust but still fast cueing mechanisms can be used. The investigation of such cueing mechanisms remains an open research challenge.

6 Conclusion

We proposed a Real-Time task scheduling framework with image resizing for criticality-based machine perception. We made the argument that resizing input images (to enable the use of lighter inference models) is a powerful tool to manipulate

resource demand for objects of different criticality. We applied empirical optimizations to adapt the framework to the imperfect and noisy segmentation results generated using a real LiDAR sensor. In addition to resizing, we allowed the scheduler to merge nearby segments into one consolidated segment to prevent over-segmentation. We designed a new scheduling algorithm and evaluated our framework using extensive experiments on a large-scale real-world autonomous driving dataset, the Waymo Open Dataset. The evaluation demonstrated substantial improvements in effectiveness and efficiency over the state of the art. We conclude that the use of LiDARs as ranging sensors is a viable approach in practice for segmenting images into smaller regions of different criticality in order to enable the implementation of criticality-based machine perception. Specifically, we explored a solution where resources are allocated differently at a sub-frame level by virtue of LiDAR-based segment consolidation and resizing in a manner that reflects the criticality of objects in the corresponding parts of the scene. The paper therefore confirms the practical feasibility of implementing criticality-based machine perception using sub-frame resizing techniques.

Acknowledgements This research was sponsored in part by DARPA award W911NF-17-C-0099 and the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196.

References

Abdelzaher T, Ayanian N, Basar T, Diggavi S, Diesner J, Ganesan D, Govindan R, Jha S, Lepoint T, Marlin B, Nahrstedt K, Nicol D, Rajkumar R, Russell S, Seshia S, Sha F, Shenoy P, Srivastava M, Sukhatme G, Swami A, Tabuada P, Towsley D, Vaidya N, Veeravalli V (2018) Toward an internet of battlefield things: a resilience perspective. Computer 51(11):24–36

Alcon M, Tabani H, Kosmidis L, Mezzetti E, Abella J, Cazorla FJ (2020) Timing of autonomous driving software: problem analysis and prospects for future solutions. In: 2020 IEEE real-time and embedded technology and applications symposium (RTAS), pp 267–280

Bamburry D (2015) Drones: designed for product delivery. Des Manag Rev 26(1):40–48

Bar-Shalom Y, Tse E (1975) Tracking in a cluttered environment with probabilistic data association. Automatica 11(5):451–460

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

Bhattacharya S, Lane ND (2016) Sparsification and separation of deep learning layers for constrained resource inference on wearables. In: Proceedings of the 14th ACM conference on embedded network sensor systems CD-ROM. SenSys '16. Association for Computing Machinery, New York, NY, USA, pp 176–189

Bogoslavskyi I, Stachniss C (2017) Efficient online segmentation for sparse 3d laser scans. PFG J Photogramm Remote Sens Geoinf Sci 85:41–52

Chen J, Yu S, Tabish R, Bansal A, Liu S, Abdelzaher T, Sha L (2021) Lidar cluster first and camera inference later: a new perspective towards autonomous driving. arXiv preprint arXiv:2111.09799

CUDA Concurrent Kernel Execution. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#concurrent-kernel-execution

Dai X, Yin H, Jha NK (2020) Incremental learning using a grow-and-prune paradigm with efficient neural networks. IEEE Trans Emerg Top Comput. https://doi.org/10.1109/TETC.2020.3037052

Denton E, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation

Floreano D, Wood RJ (2015) Science, technology and the future of small autonomous drones. Nature 521:460–466

Han S, Mao H, Dally W.J (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding

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

Hu Y, Liu S, Abdelzaher T, Wigness M, David P (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

Jin Q, Yang L, Liao Z (2020) Adabits: neural network quantization with adaptive bit-widths. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)

Jung S, Son C, Lee S, Son J, Han J-J, Kwak Y, Hwang S.J, Choi C (2019) Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)

Kellerer H, Pferschy U, Pisinger D (2004) The multiple-choice knapsack problem. Springer, Berlin, pp 317–347

Kim J-E, Bradford R, Shao Z (2020a) Anytimenet: controlling time-quality tradeoffs in deep neural network architectures. In: 2020 design, automation test in Europe conference exhibition (DATE), pp 945–950

Kim J-E, Bradford R, Yoon M-K, Shao Z (2020b) Abc: abstract prediction before concreteness. In: 2020 design, automation test in Europe conference exhibition (DATE), pp 1103–1108

Lin T-Y, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollar P, Zitnick L (2014) Microsoft coco: common objects in context. In: European conference on computer vision (ECCV)

Lin S-C, Zhang Y, Hsu C-H, Skach M, Haque ME, Tang L, Mars J (2018) The architectural implications of autonomous driving: constraints and acceleration. SIGPLAN Not 53(2):751–766

Liu S, Yao S, Fu X, Tabish R, Yu S, Bansal A, Yun H, Sha L, Abdelzaher T (2020) On removing algorithmic priority inversion from mission-critical machine inference pipelines. In: 2020 IEEE real-time systems symposium (RTSS), pp 319–332

Liu S, Yao S, Fu X, Shao H, Tabish R, Yu S, Bansal A, Yun H, Sha L, Abdelzaher T (2021) Real-time task scheduling for machine perception in intelligent cyber-physical systems. IEEE Trans Comput. https://doi.org/10.1109/TC.2021.3106496

Milioto A, Vizzo I, Behley J, Stachniss C (2019) Rangenet++: fast and accurate lidar semantic segmentation. In: 2019 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, pp 4213–4220

Minnehan B, Savakis A (2019) Cascaded projection: end-to-end network compression and acceleration. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)

Olmedo IS, Capodieci N, Martinez JL, Marongiu A, Bertogna M (2020) Dissecting the cuda scheduling hierarchy: a performance and predictability perspective. In: 2020 IEEE real-time and embedded technology and applications symposium (RTAS), pp 213–225

Otterness N, Anderson JH (2020) Amd gpus as an alternative to nvidia for supporting real-time workloads. 32nd Euromicro conference on real-time systems (ECRTS 2020), vol 165. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp 10–11023

Otterness N, Yang M, Rust S, Park E, Anderson JH, Smith FD, Berg A, Wang S (2017) An evaluation of the nvidia tx1 for supporting real-time computer-vision workloads. In: 2017 IEEE real-time and embedded technology and applications symposium (RTAS), pp 353–364

Sinha P, Zoltners AA (1979) The multiple-choice knapsack problem. Oper Res 27(3):503–515

Sun P, Kretzschmar H, Dotiwalla X, Chouard A, Patnaik V, Tsui P, Guo J, Zhou Y, Chai Y, Caine B, Vasudevan V, Han W, Ngiam J, Zhao H, Timofeev A, Ettinger S, Krivokon M, Gao A, Joshi A, Zhang Y, Shlens J, Chen Z, Anguelov D (2020) Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)

Wang Y, Xu C, You S, Tao D, Xu C (2016) Cnnpack: packing convolutional neural networks in the frequency domain. In: Lee D, Sugiyama M, Luxburg U, Guyon I, Garnett R (eds) Advances in neural information processing systems, vol 29. Curran Associates, Inc

Yang M, Otterness N, Amert T, Bakita J, Anderson JH, Smith FD (2018) Avoiding pitfalls when using nvidia gpus for real-time tasks in autonomous systems. In: Altmeyer S (ed) 30th Euromicro conference on real-time systems (ECRTS 2018), vol 106. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp 20–12021

Yang M, Wang S, Bakita J, Vu T, Smith FD, Anderson JH, Frahm J-M (2019) Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: addressing an industrial challenge. In: 2019 IEEE real-time and embedded technology and applications symposium (RTAS), pp 305–317

Yao S, Zhao Y, Zhang A, Su L, Abdelzaher T (2017) Deepiot: compressing deep neural network structures for sensing systems with a compressor-critic framework. In: Proceedings of the 15th ACM conference on embedded network sensor systems. SenSys '17. Association for Computing Machinery, New York, NY, USA

Yao S, Zhao Y, Zhang A, Hu S, Shao H, Zhang C, Su L, Abdelzaher T (2018a) Deep learning for the internet of things. Computer 51(5):32–41

Yao S, Zhao Y, Shao H, Liu S, Liu D, Su L, Abdelzaher T (2018b) 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. SenSys '18. Association for Computing Machinery, New York, NY, USA, pp 278–291

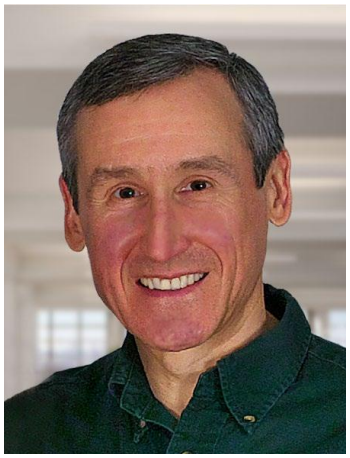
Yao S, Hao Y, Zhao Y, Shao H, Liu D, Liu S, Wang T, Li J, Abdelzaher T (2020) Scheduling real-time deep learning services as imprecise computations. In: 2020 IEEE 26th international conference on embedded and real-time computing systems and applications (RTCSA), pp 1–10

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 Ph.D. 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).

Shengzhong Liu is currently a postdoc research associate at the University of Illinois at Urbana-Champaign (UIUC). He received his Ph.D. from UIUC in 2020. His current research interests include machine learning for Internet of Things (IoT) and Cyber-Physical Systems (CPS), intelligent real-time systems, deep sensor fusion, and social network analysis.

Tarek Abdelzaher (Ph.D., UMich, 1999) is a Sohaib and Sara Abbasi Professor of Computer Science and a Willett Faculty Scholar at the University of Illinois, with over 300 refereed publications in Real-time Computing, Distributed Systems, Sensor Networks, and IoT, and an H-index of 94. He served as Editor-in-Chief of the Journal of Real-Time Systems for 15 years, and as an Associate Editor of multiple journals including IEEE TMC, IEEE TPDS, ACM ToSN, ACM TIoT, and ACM ToIT. He also chaired several top conferences in his field, including RTSS, Sensys, Infocom, ICDCS, IPSN, RTAS, DCoSS, and ICAC. 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.

Maggie Wigness 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.

Authors and Affiliations

Yigong Hu[1] · Shengzhong Liu[1] · Tarek Abdelzaher[1] · Maggie Wigness[2] · Philip David[2]

Shengzhong Liu
sl29@illinois.edu

Tarek Abdelzaher
zaher@illinois.edu

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

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

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

[2] US DEVCOM Army Research Laboratory, Adelphi, USA