Real-Time Task Scheduling for Machine Perception in Intelligent Cyber-Physical Systems

Shengzhong Liu, Shuochao Yao, Xinzhe Fu, Huajie Shao, Rohan Tabish, *Student Member, IEEE*, Simon Yu, Ayoosh Bansal, Heechul Yun, Lui Sha, *Fellow, IEEE*, and Tarek Abdelzaher, *Fellow, IEEE*

Abstract—This paper explores *criticality-based real-time scheduling* of neural-network-based machine inference pipelines in cyber-physical systems (CPS) to mitigate the effect of algorithmic priority inversion. We specifically focus on the perception subsystem, an important subsystem feeding other components (e.g., planning and control). In general, priority inversion occurs in real-time systems when computations that are of lower priority are performed together with or ahead of those that are of higher priority. In current machine perception software, significant priority inversion occurs because *resource allocation* to the underlying neural network models does not differentiate between critical and less critical data within a scene. To remedy this problem, in recent work, we proposed an architecture to partition the input data into regions of different criticality, then formulated a utility-based optimization problem to batch and schedule their processing in a manner that maximizes confidence in perception results, subject to criticality-based time constraints. This journal extension matures the work in several directions: (i) We extend confidence maximization to a generalized utility optimization formulation that accounts for criticality in the utility function itself, offering finer-grained control over resource allocation within the perception pipeline; (ii) we further instantiate and compare two different criticality metrics (distance-based and relative velocity-based) to understand their relative advantages; and (iii) we explore the limitations of the approach, specifically how inaccuracies in criticality-based attention cueing affect performance. All experiments are conducted on the NVIDIA Jetson AGX Xavier platform with a real-world driving dataset.

Index Terms—Real-time scheduling, algorithmic priority inversion, cyber-physical systems (CPS), machine intelligence

1 Introduction

OUR recent work [1] suggests that perception in modern machine inference pipelines in intelligent cyber-physical systems (e.g., autonomous cars) suffers from algorithmic priority inversion; all parts of a scene are allocated the same processing priority. This is in contrast to directing focus more to selected elements of the scene that are deemed more *critical*. To remedy the above problem, we proposed a utility-based scheduling framework to prioritize processing of parts of the scene over others, which we call *attention cueing*. By "attention", we do not mean AI mechanisms that increase the *logical* weights of certain features over others [2] (as such mechanisms do not change the amount of resources expended on processing the weighted features). Rather, we refer to changes in *actual resource allocation* by scheduling the perception pipeline to process certain parts of the input first. In our previous work, attention cueing was driven by maximizing confidence in scene understanding subject to time constraints [1].

This journal extension continues the above investigation by (i) generalizing the proposed resource allocation framework by incorporating criticality into the utility function itself (as opposed to merely reflecting it in time constraints), (ii) comparing the effects of different criticality assignments, and (iii) exploring the implications of imperfect attention cueing on performance. Incorporating criticality into the utility function (not only the constraints) allows it to serve as an *optimization criterion* as opposed to merely a *satisficing metric*. Thus, as we show in the evaluation section, improvements are possible in scheduling performance.

The general framework proposed in this paper allows expressing arbitrary application-specific criticality metrics (assigned to parts of input frames) to direct resource allocation. We implement and experiment with two instantiations of such criticality metrics: distance-based criticality and relative velocity-based criticality. They are based on different arguments inspired by path planning in autonomous driving. Comparing the results to our conference version [1], we demonstrate improvements in both schedulability and object classification accuracy thanks to resource savings

- *Shengzhong Liu, Rohan Tabish, Ayoosh Bansal, Lui Sha, and Tarek Abdelzaher are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: {sl29, rtabish, ayooshb2, lrs, zaher}@illinois.edu.*
- *Shuochao Yao is with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA. E-mail: shuochao@gmu.edu.*
- *Xinzhe Fu is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. E-mail: xinzhe@mit.edu.*
- *Huajie Shao is with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23185 USA. E-mail: hshao@wm.edu.*
- *Simon Yu is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. E-mail: jundayu2@illinois.edu.*
- *Heechul Yun is with the Department of Eclectrical Engineering and Computer Science, The University of Kansas, Lawrence, KS 66045 USA. E-mail: heechul.yun@ku.edu.*

Manuscript received 23 Nov. 2020; revised 20 May 2021; accepted 15 Aug. 2021. Date of publication 20 Aug. 2021; date of current version 11 July 2022. (Corresponding author: Shengzhong Liu.) Recommended for acceptance by G. C. Sirakoulis. Digital Object Identifier no. 10.1109/TC.2021.3106496

attained by reducing time spent on processing backgrounds and far away objects. We then explore the impact of imperfections in attention cueing on algorithm performance and show that these imperfections affect the perception pipeline differently for different choices of neural network architecture, demonstrating a trade-off between execution efficiency and robustness to inaccuracies.

We focus on the perception subsystem because perception, besides being a key component in enabling system autonomy, is also a major efficiency bottleneck that accounts for a considerable fraction of resource consumption [3]. The work improves the design of mission-critical cyber-physical systems that need to perceive their environment in real time (using neural networks), such as self-driving vehicles [4], autonomous delivery drones [5], and military defense systems [6].

We note that, in perception pipelines, attention cueing may appear to pose a "chicken and egg" problem. In order to attend to more critical parts of the scene first, one needs to identify regions of higher criticality. However, in order to break down the scene into regions of different criticality, one needs to have inspected the scene already. This circular dependency is broken, in this paper, by using a second sensor whose purpose is simply to identify places that deserve a better look, much the same way that sound might cue a person to look in one direction or the other. Specifically, we suggest to use a *ranging sensor*. Ranging identifies depth and therefore depth discontinuities, allowing one to quickly isolate foreground objects from backgrounds [1]. LiDAR offers reliable depth information for individual pixels under normal weather conditions. Foreground objects in the scene can thus be approximately localized depending on depth information [7], offering a basis for prioritization (before any neural network processing begins). We can then attend to the nearest objects first, or the most rapidly approaching objects first, roughly translating into distance-based and velocity-based criticality resource allocation policies, respectively. Attention cueing within the perception subsystem can be viewed as an optimization to mission-critical perception functions. Our work does not obviate safety-critical (emergency) overrides to handle corner cases where safety becomes in jeopardy.

We implement the architecture on an NVIDIA Jetson AGX Xavier platform, and do a performance evaluation using real video traces collected from autonomous vehicles. The results show that prioritization, by dropping inconsequential regions (e.g., sky, backgrounds, etc), significantly improves the average quality of machine inference, while nearly eliminating deadline misses, compared to a set of state-of-the-art baselines under the same settings.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 is a conceptual overview of the proposed architecture. Section 4 describes the scheduling algorithms developed. We introduce the criticality-based weight design in Section 5. System evaluation is presented in Section 6. We provide an open discussion about other potential criticality designs in Section 7. The paper concludes with Section 8.

2 Related Work

The work is motivated by the large expansion of modern cyber-physical systems (CPS) research into areas of machine intelligence and autonomy to enable progressively broader categories of tomorrow's mission-critical applications [8]. Current machine learning software has been very successful at producing run-time inference algorithms that approach or exceed capabilities of human perception [9]. Of particular promise have been recent advances in neural networks [10]. However, mainstream deep neural network inference algorithms are not designed explicitly with *timing and criticality constraints* of cyber-physical systems in mind, generating a need to refactor modern neural network software.

In the broader neural network research literature, much work was done on deep model compression and acceleration [11]. Examples include parameter quantization [12], weight virtualization [13], node pruning [14], and low-rank projection [15]. In addition, several advances have focused on offloading parts of on-device neural network inference to the cloud [16]. We complement that thread by introducing the notion of prioritization into the AI workflow. We exploit physical aspects of the platform and the application to enable additional reductions in cost while improving predictability, and timeliness.

Recent efforts on AI-empowered real-time systems addressed CPU/GPU scheduling for pipelined machine inference [17], resource and energy management [18], [19], and communication and collaboration protocol design [20]. Autonomous driving emerged as a flagship application motivating AI-empowered real-time system design [21]. Extensive hardware and software evaluations have been performed to understand and reduce the end-to-end pipeline delays [3], [22], as well as improve the robustness of hardware accelerators [23]. Recent papers refactored deep neural networks to satisfy dynamic execution-time constraints during inference [24], [25], [26] with heterogeneous execution patterns. For example, Bateni *et al.* [27] applied a combination of different layer-wise network approximation techniques to meet target deadlines. Lee *et al.* [24] introduced dynamic subnetwork construction for DNNs (where the subnetwork with best performance that meets time constraints is selected at runtime). Liu *et al.* [1] were the first to propose a prioritization pipeline to mitigate the priority inversion problem in machine perception subsystems. We further extend it by offering a generalized criticality assignment mechanism to to achieve a better tradeoff between prioritized responses (in terms of both time and quality) to critical segments of input and the average model inference quality. We also investigate limitations of the approach.

3 System Architecture

Consider an intelligent cyber-physical system (i.e., the observer) equipped with a camera that observes its physical environment, a neural network that processes the observations, and a control unit that must react in real time. In the conventional machine perception pipeline, input data frames captured by sensors are processed sequentially as intact units. Network execution is typically non-preemptive. It considers one frame at a time, producing an output on each frame before the next frame is handled.

Unfortunately, the data frames captured by modern sensors (e.g., colored camera images) carry information of different degrees of criticality in every frame. Data of different

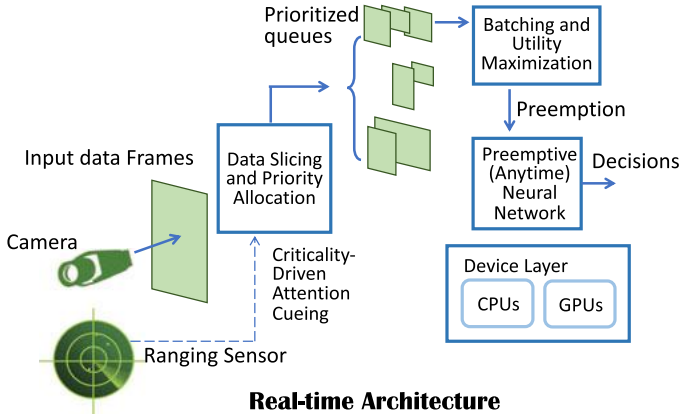


Fig. 1. A Real-time Machine Perception Pipeline (originally suggested in [1]).

degrees of criticality may favor a different processing latency within the perception subsystem. For example, parts of the image containing close objects may require a faster response compared to parts containing distant objects, while static background regions with no objects may not need processing at all (at least not in every frame). To accommodate these differences, we consider the perception architecture shown in Fig. 1. It features three key components:

- *The data slicing and priority allocation module:* This module breaks up newly arriving frames into smaller regions with different degrees of criticalty based on simple heuristics.
- *The "anytime" neural network:* This neural network implements an imprecise computation model that yields partial utility from the partially completed computation. It allows saving resources on processing less critical regions by the perception subsystem.
- *The batching and utility maximization module:* This module sits between the data slicing on one end and the neural network on the other. With data regions broken by priority, it decides which regions to pass to the neural network for processing. Since multiple regions may be queued for processing, it also decides how best to benefit from batching (that improves processing efficiency).

For completeness, below we first describe all of the above components of the observer, respectively. We then detail the batching and utility maximization algorithm.

3.1 Data Slicing and Priority Allocation

This module breaks up input data frames into regions of different criticality. In this paper, we consider criticality assignment based on output of a *ranging sensor*. The current autonomous vehicle industry has an ongoing debate on the best way to implement the ranging sensor [28], [29], [30], [31].[1] Much of the industry, including Waymo, GM, Nuro, Uber, Aurora, Zoox, Argo, and Lyft, have invested in LiDAR. Apple announced (in late 2020) investing in an autonomous car that may use the company's own LiDAR technology as well. On the opposition side, most prominently, Elon Musk's Tesla openly opposes LiDAR on several grounds, including cost. We do not take a position on this debate. While in the rest of the paper, we shall call our ranging sensor a LiDAR, we acknowledge that one can alternatively reconstruct depth using other means such as stereo cameras (similar to the human vision system), although sometimes challenging [30], and even thermal cameras [32]. On the advantage side, LiDAR directly estimates distance without the need for synchronization or neural models [31]. It is therefore potentially more reliable in avoiding accidents caused by failed distance prediction under normal weather conditions [33]. LiDAR, however, is effectively blind in bad weather conditions. Under these conditions, a different ranging sensor should be used instead (or the driver should switch to a manual mode). Finally, LiDAR point cloud based object localization techniques have been proposed in recent literature [7]. They provide a fast (i.e., over 200 Hz) ranging and object localization capability that do not require GPU processing.

The data slicing and priority allocation module needs only to identify *approximate* locations of potential objects from differences in depth, but does not need to recognize object categories or determine exact outlines. A padding margin can be added around the identified depth-based foreground regions to allow for inaccuracies in capturing object boundaries. We investigate the impact of such inaccuracies on algorithm performance (where larger inaccuracies call for larger margin sizes, leading to efficiency loss). The padded regions have rectangular shapes. They constitute (approximate) *bounding boxes* for the respective objects. The development and evaluation of specific object localization algorithms is outside the scope of this paper, as they have akready been addressed in prior work [7].

Each identified rectangular region, i, is associated with a criticality weight w_i. In this paper, we compare two instantiations of object criticality, namely, *distance-based criticality* and *relative velocity-based criticality*. Besides foreground objects identified by depth irregularities (as mentioned above), we can view the rest of the frame as a special segment of background priority, for example by allowing AI-based perception algorithms to look at entire frames (including the background) at a lower frequency. We can control the scheduling of that special segment by manipulating its associated wight function, w_i. This modification will also allow the perception subsystem to process static parts of the scene and perform other functions such computing geometric relations between objects.

3.2 The Anytime Neural Network

A perfect *anytime* algorithm is one that can be terminated at any point, yielding utility that monotonically increases with the amount of processing performed. Our neural network approximates that model. Specifically, it implements an *imprecise computation* model [35] that provides usable and approximate partial results. In an imprecise computation model, processing consists of two parts: a *mandatory part* and an *optional part*. The optional part, or a portion thereof, can be skipped to conserve resources. When the optional part is skipped, the task is said to produce an *imprecise* (i.e., approximate) result.

1. https://www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race/

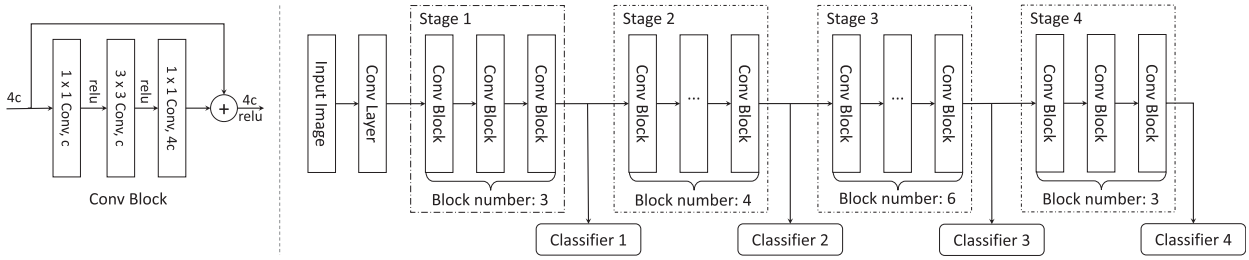


Fig. 2. ResNet [34] architecture with 4 stages and 50 layers.

Deep neural networks (e.g., image recognition models [34]) are a concatenation of many layers that can be divided into several stages, as we show in Fig. 2. Ordinarily, an output layer is used at the end to convert features computed by earlier layers into the output value (e.g., an object classification). Prior work has shown, however, that other output layers can be forked off of intermediate stages producing meaningful albeit imprecise outputs based on features computed up to that point [36]. Neural network inference can be divided into a mandatory part and optional parts. The quality of outputs increases when the network executes more optional parts.

Modeling neural networks as imprecise computations allows the scheduling algorithm to stop the computation for a task when its prediction quality is sufficiently high, or terminate low-criticality tasks earlier (with more approximate results) in favor of higher criticality tasks, facilitating resource savings. The approach needs to be used with care. Clearly, loss of fidelity can cause safety problems. Note, however, that an obstacle viewed, say, for 5 seconds at a frame rate of 30 frames/sec will be classified 150 times. As a result, per-frame classification accuracy can be a bit more relaxed, especially for objects that are a safe distance away (i.e., lower criticality), which is precisely the insight being exploited here. We adopt the algorithm proposed by Yao *et al.* in RDeepSense [37] to estimate expected confidence in outputs of future neural network stages. It allows an accurate estimation of confidence at the output of a neural network stage before the stage is executed, allowing us to compute the expected utility of each stage upfront.

3.3 Batching and Utility Maximization

This module decides the schedule of processing of all regions identified by the data slicing and prioritization module and that pass de-duplication. Below, we describe how the batching and utility maximization module schedules the tasks that process the different bounding boxes.

4 The Scheduling Problem

In this section, we describe our task execution model and formulate the scheduling problem studied in this paper. We then derive a near-optimal solution.

4.1 The Execution Model

As alluded to earlier, the scheduled tasks in our system constitute the execution of multi-layer deep neural networks (e.g., ResNet [34]), each processing a different input data region (a bounding box). As shown in Fig. 2, tasks are broken into stages. Each stage includes multiple neural network layers. The unit of scheduling is a single stage. A task arrives when a new object is detected by the ranging sensor (e.g., LiDAR) giving rise to a corresponding new bounding box in the camera scene. Let the arrival time of task τ_i be denoted by a_i. A deadline $d_i > a_i$, is assigned by the data slicing and priority assignment module denoting the time by which the task must be processed. Both a_i and d_i are a multiple of frame inter-arrival time, H. We do not pose periodicity assumptions on object arrival times and deadlines. No task can be executed after its deadline. Future object sizes, arrival times, and deadlines are unknown, which makes the scheduling problem an *online decision problem*.

4.1.1 Batching

Stages of the neural networks are executed on a GPU. One way of exploiting their computation capabilities is to execute the same kernel on all GPU cores. This means that we can run *different* tasks concurrently on the GPU as long as we run the *same kernel* on all GPU cores. We call the assembly of such concurrently executable task sets, *batching*. Running the same kernel on all GPU cores means that we can only batch tasks if both of the following applies: (i) they are executing *the same neural network stage* and (ii) they *run on the same size inputs*. Batching is advantageous because it allows us to better utilize the GPU, so we want to take advantage of it in scheduling. To increase batching opportunities, we limit the size of possible bounding boxes after slicing to a finite set of options. For a given size k, at most $B^{(k)}$ tasks can be batched together before overloading GPU capacity. We call it the *batching limit* for the corresponding input size.

4.1.2 Imprecise Computations

Let the number of stages in the neural network for task τ_i be denoted by L_i. We call the first stage *mandatory* and call the remaining stages *optional*. Following a recent study [38], tasks are written such that they can return a classification result once the mandatory stage is executed. This result then improves with the execution of each optional stage. Earlier work presented an approach to estimate the expected confidence in correctness of results of future stages, ahead of executing these stages [37]. This estimation offers a basis for assessing utility of future task stage execution. We denote by $R_{i,j}$ the predicted confidence in correctness of task τ_i after executing $j \leq L_i$ stages. This quantity can be computed as proposed by Yao *et al.* [37]. Note that, this confidence can be different among tasks (depending in part on input size), but it is computable, non-decreasing, and concave with respect to the network stage [37]. We denote by $\mathcal{T}(t)$ the set of *current tasks* at scheduling period t. A task, τ_i, is called *current* at period t, if $a_i \leq t < d_i$, and the

task has not yet completed its last stage, L_i. For task τ_i of input size, k, the execution time of the jth stage is denoted by $e_{j,b}^{(k)}$, where b is the number of batched tasks during execution. We obtain the stage execution time of each batching size through offline profiling.

4.2 Problem Formulation

The problem addressed in this paper, which we call the *Weighted Batched Online Object-recognition Scheduling with Imprecise Computation (Weighted BOOSIC)* problem, is to decide on the number of stages $l_i \leq L_i$ to execute for each task τ_i and to schedule the batched execution of those task stages on the GPU such that the total weighted utility, $\sum_i w_i R_{i,l_i}$, of executed tasks is maximized, and batching constraints are met. In this formulation, the weighted utility of executing l_i stages of task, τ_i, *by the deadline* is given by $w_i R_{i,l_i}$, where R_{i,l_i} is the confidence in correctness of task output (computed after l_i stages) and w_i is a weight that depends on task criticality. In summary:

The Weighted BOOSIC Problem. With online task arrivals, the optimization objective is to derive a schedule x to maximize the weighted sum of task utilities, where the weight is defined as the object criticality. The schedule decides three outputs: task stage execution order, task execution depth (i.e., number of stages to execute of each task), and task batching (which tasks to execute together). Specifically, for each scheduling period t, we use $x_t(i,j) \in \{0,1\}$ as an indicator variable to denote whether the jth stage of task τ_i is executed. Besides, we use P to denote a batch of tasks, where $|P|$ denotes the number of tasks being batched. The mathematical formulation is

$$\begin{aligned} &\textit{Weighted BOOSIC}: \\ &\max_{x} \sum_{t}\sum_{i} w_i x_t(i,j)\big(R_{i,j} - R_{i,j-1}\big) \\ &\textbf{s.t. } x_t(i,j) \in \{0,1\},\ \sum_{t=1}^{T} x_t(i,j) \leq 1, \quad \forall i,j \end{aligned} \tag{1}$$

$$x_t(i,j) = 0, \quad \forall t \notin [a_i, d_i),\ \forall i,j \tag{2}$$

$$\begin{aligned} &\sum_{t'=1}^{t-1} x_{t'}(i,j-1) - x_t(i,j) \geq 0, \\ &\forall i,j > 1, t > 1 \end{aligned} \tag{3}$$

$$\begin{aligned} &s_i = s_{i'} = k,\ l_i = l_{i'},\ |P| \leq b_k, \\ &\forall i \in P,\ i' \in P,\ \exists k \in \mathcal{S}. \end{aligned} \tag{4}$$

The following set of constraints have to be satisfied: (1) Each network stage for each task can only be executed once; (2) No task can be executed after its deadline; (3) The execution of different stages of the same task must satisfy their precedence constraints; (4) Only tasks with the same (image size, network stage) can be batched, and the number of batched tasks can not exceed the batching constraint of their image size. Only one batch can be executed on the GPU at any time. However, multiple batches can be executed sequentially in one period, as long as the sum of their execution times does not exceed the period length, H.

4.3 An Online Scheduling Framework

Our scheduler is invoked upon frame arrivals, which is once every H units of time. We thus call H the *scheduling period*. We assume that all task stage execution times are multiples of some basic time unit δ, thereby allowing us to express H by an integer value. We further call the problem of scheduling current tasks within the period between successive frame arrivals, the *local (weighted) scheduling problem*:

Definition 1 (The Local Weighted BOOSIC Problem). *Given the set of current tasks, $\mathcal{T}(t)$, within scheduling period, t, the local weighted BOOSIC scheduling problem seeks to maximize the total weighted utility gained within this scheduling period only.*

Liu *et al.* proved in [1] that an online scheduling algorithm that optimally solves the local scheduling problem can have a good competitive ratio relative to a Clairvoyant optimal scheduler (that has full knowledge of all future task arrivals). They correspondingly propose a locally optimal dynamic programming algorithm (named *OnlineDP*). However, it is too slow to run in real time, so we focus on a greedy algorithm that is computationally efficient and shows similar performance as OnlineDP in experiments.

Algorithm 1. Local Weighted Greedy Scheduling

Input: Available task set $\mathcal{T}(t)$ with their weighted utilities, the batch limit $B^{(k)}$ for each size k.
Output: Local task schedule x_t

while *until the end of the period* **do**
  for $k = 1, \ldots, K$ **do**
    for $j = 1, \ldots, L^{(k)}$ **do**
      $\mathcal{T}_{k,j}(t) :=$ available tasks of size k at stage j.
      if $|\mathcal{T}_{k,j}(t)| \leq B^{(k)}$ **then**
        $U_{k,j}(t) :=$ weighted utility of $\mathcal{T}_{k,j}(t)$,
        $\tilde{\mathcal{T}}_{k,j}(t) := \mathcal{T}_{k,j}(t)$.
      end
      else
        $\tilde{\mathcal{T}}_{k,j}(t)$: = $B^{(k)}$ tasks with the max weighted utility in $\mathcal{T}_{k,j}(t)$,
        $U_{k,j}(t)$: = weighted utility of tasks in $\tilde{\mathcal{T}}_{k,j}(t)$.
      end
    end
  end
  Select the pair (k,j) and execute the tasks in $\tilde{\mathcal{T}}_{k,j}(t)$ with the maximum weighted utility $U_{k,j}(t)$.
end
return x_t

The scheduling decision is made stage-wise, which means that the selected tasks will execute for only one stage before making the next scheduling decision. The greedy online scheduling algorithm solves the local BOOSIC scheduling problem following a simple greedy selection rule. In order to maximize the benefit of task batching, instead of selecting one task with the maximum marginal utility (for its next stage), the greedy algorithm executes the eligible[2]

2. By *eligible* we mean: 1) The tasks belonging to the batch all have the same image size and (valid) network stage; 2) No task has passed its deadline; 3) The number of batched tasks is no greater than the batching limit for their image size; 4) The batch can finish executing the next stage before the end of current period.

batch with the maximum utility next. The pseudo-code of the greedy scheduling algorithm is shown in Algorithm 1. The greedy scheduling algorithm is simple to implement and has a very low computational overhead. We show that it achieves a comparable performance to the optimal algorithm in practice.

5 Criticality-Based Weight Design

The scheduling objective defined in the vanilla BOOSIC problem [1] aims to optimize the overall confidence in model predictions. It allocates computational resources to tasks whose execution attains the largest increase in confidence. Tasks are split into two groups (critical or non-critical) by thresholding on object distance. Critical tasks are assigned a relatively high but fixed weight.

This paper generalizes the above by considering a fine-grained task-specific criticality weight w_i for each task that is multiplied by confidence. If we set the weight factor $w_i = 1$ for each task, then the algorithm always prioritizes tasks with high confidence increase. On the contrary, if we set the predictive confidence $x_i = 1$ for each task, then the task execution order strictly complies with task criticality. By considering both factors, we achieve more effective utilization on limited computation resources: We allocate resources preferentially to more critical tasks that also observe a substantial increase in prediction confidence.

We instantiate two different mechanisms for computing weight, w_i, in this paper: distance-based criticality, and relative velocity-based criticality. They are discussed below.

5.1 Distance-Based Criticality

As a straightforward instantiation, we first propose and use distance-based criticality. It is tempting to assume that (in a purely distance-based criticality assignment algorithm) closer objects should monotonically receive a higher priority because they induce a higher risk of future collisions. As suggested by one of our anonymous reviewers, however, this reasoning is flawed. Objects closer than a certain minimum threshold, l_{min}, might already be too close to comfortably avoid. It is thus imperative to account for them while they are are still sufficiently far way. In other words, the value function for determining criticality is *shifted*. Namely, assume the distance of the ith object is l_i and the maximum LiDAR detection range is l_{max}, the distance-based criticality weight is defined as

$$w_i = \begin{cases} 0, & \text{if } l_i \le l_{min}; \\ \frac{1}{\left(\frac{l_i - l_{min}}{l_{max} - l_{min}}\right)^k + \epsilon}, \ k \ge 1, & \text{if } l_i > l_{min}, \end{cases} \tag{5}$$

where ϵ is a small positive term for stabilization. The threshold l_{min} is the minimal safe object distance, defined as

$$l_{min} = v \cdot H\delta + \frac{v^2}{2 \cdot a}, \tag{6}$$

where v is the current velocity of the AV, and a is the largest acceleration of the AV when it makes a hard brake. Equation (6) computes the minimum safe distance l_{min}, in a velocity-dependent fashion, for the vehicle to avoid an obstacle. Below that distance, a safety-critical override (e.g., a collision avoidance system) should immediately intervene and stop the car or reduce speed (to increase l_{min}). Note that, if the car is stopped, l_{min} is zero. We henceforth call l_{min} a (distance) *shift point*, since the weight assignment, w_i is effectively shifted by l_{min} to focus on more distant objects.

One limitation of the above design is that it does not account for the velocity of the other objects. In general, an object that is decelerating might be more of a concern than one that is accelerating (away from the AV), even if both are presently the same distance away. Below, we describe a modification of the above algorithm that accounts for relative velocity.

5.2 Relative Velocity-Based Criticality

We can estimate relative velocity information by computing distance change between consecutive frames. We need an efficient object tracking module to calculate relative velocity of surrounding objects. We follow the SORT algorithm proposed by Bewley *et al.* [39] for object tracking. It adopts a tracking-by-detection strategy to track multiple objects in parallel. At each frame, we try to map each bounding box to an existing object track through a data association algorithm (the Hungarian algorithm [40] in our implementation), based on the intersection-over-union (IOU) matrix between new bounding boxes and predicted locations for object tracks. The assumption is that, if a new bounding box is highly overlapped with the projected location of a previous object according to the motion model extracted from its past trajectory, then we believe they belong to the same object. No semantic information (i.e., category of the object) is needed during the mapping. The relative velocity is

$$\tilde{v}_i = \frac{l_i' - l_i}{H\delta}, \tag{7}$$

where l_i and l_i' denote the distance of object i in current period and the previous period respectively. A positive value means the object is approaching while a negative value means the object is moving away. We further remove object mappings that lead to an abnormally high relative velocity. We correspondingly define the object deadline as

$$d_i = \begin{cases} d_{max} & \text{if } \tilde{v}_i \le 0 \text{ or } \frac{l_i}{\tilde{v}_i} > d_{max}; \\ \frac{l_i}{\tilde{v}_i} & \text{if } 0 < \frac{l_i}{\tilde{v}_i} \le d_{max}, \end{cases} \tag{8}$$

where $d_{max} = \frac{l_{max}}{v_o}$ is defined as the maximum deadline. It is calculated using the maximum LiDAR range l_{max} and the observer velocity v_o. The object weight is

$$w_i = \begin{cases} 0 & \text{if } d_i \le d_{min}; \\ \frac{1}{\left(\frac{d_i - d_{min}}{d_{max} - d_{min}}\right)^k + \epsilon}, \ k \ge 1, & \text{if } d_i > d_{min}. \end{cases} \tag{9}$$

As in the previous section, we assume when an object deadline falls below the threshold d_{min}, the safety-critical system should immediately interfere and take corresponding action. We call d_{min} a (deadline) shift point.

One limitation of this mechanism is that we need at least two appearances of the same object to calculate its relative velocity. In the data association step, if the new bounding box can not be mapped to any existing tracks, then it is

considered as a potential new object. In this case, we assume the object to be static.

6 Evaluation

In this section, we verify the effectiveness and efficiency of our proposed scheduling framework by comparing it with several state-of-the-art baselines on a large-scale self-driving dataset, Waymo Open Dataset.

6.1 Experimental Setup

6.1.1 Hardware Platform

All experiments are conducted on an NVIDIA Jetson AGX Xavier SoC. It's equipped with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory. Its mode is set as MAXN with maximum CPU/GPU/memory frequency budget, and all CPU cores are turned on.

6.1.2 Dataset

Our experiment is performed on the Waymo Open Dataset [41], which is a large-scale autonomous driving dataset collected by Waymo self-driving cars in diverse geographies and conditions. It includes driving video segments of 20s each, collected by LiDARs and cameras at 10 Hz. All LiDAR and camera data are synchronized. Only the front camera data is used in our experiment. Since we do not need the added resolution, objects with size larger than 256 are down-scaled to 256 while preserving its aspect ratio. All remaining images are padded to the target size bins.

6.1.3 Neural Network

Unless otherwise stated, we use ResNet proposed by He *et al.* [34] for object classification. The network is trained on a general-purpose object detection dataset, COCO.[3] It contains 80 object classes that include those of the Waymo dataset. An advantage of ResNet is that its architecture is amenable to modification to support the imprecise computation model as noted in our prior work [38]. A disadvantage of ResNet is that it is a *classification* neural network. Thus, if a region of interest contains multiple objects of different types, ResNet will not be able to individually identify them. To avoid this problem, one can use YOLO [42]. Unlike ResNet, YOLO first detects where the individual objects are, then classifies each. With YOLO, it becomes possible to handle regions (returned by the ranging sensor) that contain multiple objects. As we show later, this flexibility increases robustness to inaccuracies in object localization. A disadvantage is that YOLO does not lend itself well to separation into mandatory and optional parts. This limitation results in a loss of efficiency because one has to execute either all stages of the YOLO neural network or nothing. For space limitations, we restrict most of the evaluation to ResNet, but offer a comparison of ResNet and YOLO in Section 6.7.

6.1.4 Scheduling Load and Evaluation Metrics

We extract the distance between objects and the autonomous vehicle (AV) from the projected LiDAR point cloud. The deadlines of object classification tasks are set as the time to collision (TTC) with the AV. To simulate different loads for the scheduling algorithms, we manually change the sampling period (i.e., frame rate) from 40 ms to 160 ms. We consider a task to miss its deadline if the scheduler fails to run the mandatory part by the deadline. Otherwise, we consider the task to return a timely but possibly imprecise result. In following evaluations, we present both *normalized accuracy* and *deadline miss rate* for compared algorithms. The normalized accuracy is defined as the ratio between achieved accuracy and the maximum accuracy when all neural network stages are executed for the object.

6.2 Compared Scheduling Algorithms

The following scheduling algorithms are compared.

- *OnlineDP*: It refers to the online scheduling algorithm proposed in [1]. The optimal local scheduling in each period is conducted by the hierarchical dynamic programming algorithm. We use it as an upper bound on *average* model accuracy, since its optimization objective is equivalent to maximizing the achieved average model accuracy with no regard to criticality.
- *Greedy-Uni*: This is a simplification of the online scheduling algorithm proposed in [1], wherein the local scheduling conducted by an unweighted but batched greedy algorithm. The task weight is uniformly set as $w_i = 1$ for all tasks. Its scheduling objective is also equivalent to (approximately) maximizing the achieved model accuracy on all objects.
- *Greedy-WeiD/WeiV*: This is the proposed online scheduling algorithm in this paper, with the local scheduling conducted by the weighted and batched greedy algorithm. Its scheduling objective is a trade-off between prioritizing execution for critical objects, and the overall model accuracy on all objects. In the following experiments, we use -WeiD to denote the distance based-criticality assignment and use -WeiV to denote the relative velocity-based criticality assignment. The default implementation neglects the shift point (i.e., sets l_{min} and d_{min} to zero). We use -SFT to denote algorithm variations with a non-zero shift point (i.e., $l_{min} \neq 0$ or $d_{min} \neq 0$).
- *Greedy-NB*: It always executes the single (task, stage) with maximal marginal utility. No batching is performed. Utility is set proportional to the achieved predictive confidence. In other words, task weight is uniformly set to $w_i = 1$, for all tasks.
- *Greedy-NB-WeiD/WeiV*: Same as *Greedy-NB*, except that the weight, w_i is calculated according to distance-based criticality (-WeiD) or velocity-based crticality (-WeiV).
- *EDF:* It always chooses the single task stage with the earliest deadline (without considering task utility). To offer immediate service, no batching is performed (in EDF and subsequent algorithms below).
- *Non-Preemptive EDF (NP-EDF):* Unlike regular EDF, this algorithm does not allow preemption. Once a task starts executing, it continues until it is finished or its deadline is reached. It is included to

3. https://cocodataset.org/#home

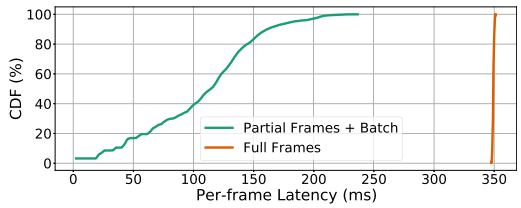


Fig. 3. Cumulative distribution of end-to-end latency on full frames between full frame inference and batched partial frame inference. The execution time for frame slicing, batching, and neural network inference are all counted.

understand the impact of allowing preemption on stage boundaries compared to not allowing it.

- *FIFO:* It runs the task with the earliest arrival time first. All stages are performed if the deadline is not violated.
- *RR:* Round-robin scheduling algorithm. Runs one stage of each task in a round-robin fashion.

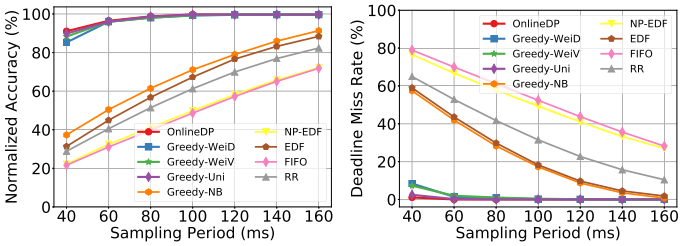
6.3 Slicing and Batching

Next, we compare the inference time for *full frames* and *batched partial frames*. In full frame processing, we directly run the neural network on camera-captured full images, whose size is 1920×1280. In *batched partial frames*, we do the slicing into bounding boxes within one frame first, and then batch execution of objects with the same size. Each frame is evaluated independently. No imprecise computation is considered. The end-to-end latency for each full frame, including both preprocessing and network inference time, is reported here. Our results show that the average latency for full frames is 350ms,[4] while the average latency for (the sum of) batched partial frames is 105ms. The cumulative distributions of frame latencies for the two methods are shown in Fig. 3. Data slicing and batching, although induce extra processing delays, can effectively reduce the end-to-end latency. This experiment quantifies the minimum amount of improvement one can expect due to slicing and batching alone. Batching is done among same-frame regions only (in FIFO order). The results already show significant reduction in response latency compared to full frame execution. The imprecise computation model, and the utility-based scheduling algorithm would further reduce the response time by implementing better prioritization and allowing cross-frame batching.

6.4 Scheduling Policy Comparisons

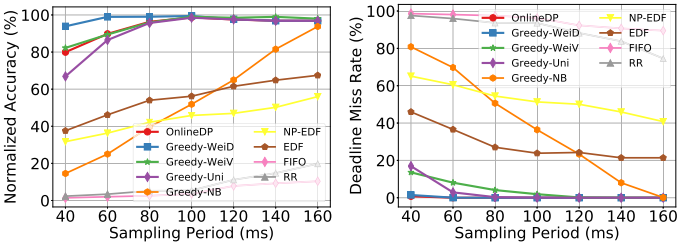
Next, we evaluate the scheduling algorithms in terms of achieved classification accuracy and deadline miss rate. We change the replayed camera frame rate to vary the load. The scheduling results are presented in Fig. 4. The batched algorithms, OnlineDP, Greedy-WeiD/WeiV and Greedy-Uni, clearly outperform all non-batched baselines with a large margin in both metrics. The improvement comes for two reasons: First, the integration of imprecise computation model makes the neural network execution partially preemptive. Second, the task batching mechanism significantly increases the task processing throughput. The deadline miss rates of OnlineDP and Greedy-WeiD/WeiV/Uni are

4. There can be small variations in full frame inference time due to the system dynamics.



(a) Normalized accuracy. (b) Deadline miss rate.

Fig. 4. Accuracy and deadline miss rate comparisons on all objects.

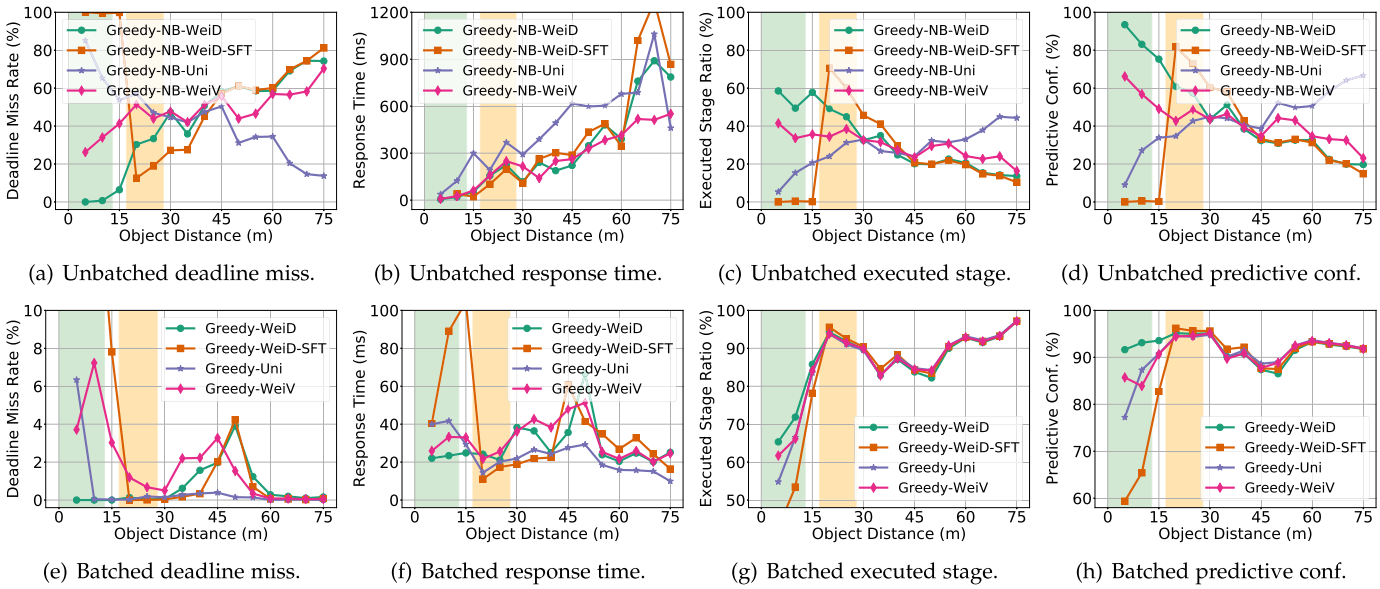


(a) Normalized accuracy of critical objects. (b) Deadline miss rate of critical objects.

Fig. 5. Accuracy and deadline miss rate comparisons on critical objects.

pretty close to 0 under any task load. However, we also find that Greedy-WeiD and Greedy-WeiV present a small degradation in both normalized accuracy and deadline miss rate when the sampling period is small (i.e., 40ms). As we will show in Fig. 5, the degradation mainly comes as a cost of prioritizing processing the critical tasks. Greedy-Uni and OnlineDP show similarly the best performance, because their scheduling objective (i.e., unweighted sum of task utilities) is equivalent to maximizing the overall accuracy. Instead, in the scheduling objective of Greedy-WeiD/WeiV, low-criticality tasks have smaller weight factors and are more likely to be skipped. Essentially, Greedy-WeiD/WeiV trade part of prediction quality on low-criticality tasks for timely and high-quality prediction on critical objects, though their definitions of criticality differ.

To better see the above trade-off, we now focus our attention on more critical objects. In this section, we first define critical objects as those whose distance is within 10m. Since such objects might already be too close (as pointed out by one of the anonymous reviewers), in the next section, we explore the impact of shifting criticality weights to favor more distant objects. Results are shown in Fig. 5. We notice that the accuracy and deadline miss rate of FIFO and RR are much worse on critical objects, because severe priority inversion occurs in these two algorithms. The deadline-driven algorithms (NP-EDF and EDF) can effectively resolve priority-inversion. However, their performance on critical objects is inferior due to the limited task processing throughput without task batching. Greedy-NB, although obtains better general performance than EDF and NP-EDF, shows worse performance on critical objects when the sampling period is short. The batched scheduling algorithms (Greedy-WeiD/WeiV/Uni and OnlineDP) show better performance for critical objects than non-batched algorithms because of the increased task processing capacity that comes with batching. A performance difference between different batched algorithm flavors appears at high load (i.e., for



(a) Unbatched deadline miss. (b) Unbatched response time. (c) Unbatched executed stage. (d) Unbatched predictive conf.

(e) Batched deadline miss. (f) Batched response time. (g) Batched executed stage. (h) Batched predictive conf.

Fig. 6. Scheduling results of greedy algorithms on objects with *different distances*, when applying distance-based weight (*WeiD*), relative velocity-based weight (*WeiV*) and uniform (*Uni*) weight. We separately report the results with and without task batching. The range highlighted with green shows the advantage of *Greedy(-NB)-WeiD*, and the range highlighted with orange shows the advantage of *Greedy(-NB)-WeiD-SFT*.

shorter sampling periods), because task batching manages to accommodate (almost) all tasks within capacity, when the sampling period is long enough. Focusing on short periods, among the batched algorithms, Greedy-WeiD shows the best performance on critical objects, with near-optimal accuracy and almost no deadline misses. Greedy-Uni and OnlineDP are inferior because their scheduling objective assigns the same importance (weight) to all tasks. Under uniform scheduling, critical objects with shorter deadlines are more likely to execute fewer neural network stages or pass their deadlines, thus receiving inferior quality (i.e., accuracy). Similarly, Greedy-WeiV does not show optimal performance on close objects (defined as critical in Fig. 5) as Greedy-WeiD, because it prioritizes the fast approaching objects over the close objects (with slow approaching velocity). The results demonstrate the trade-off involved in favoring critical tasks by Greedy-WeiD at the cost of a minor degradation in average model performance, shown in Fig. 4.

6.5 Impact of Criticality Assignment Mechanisms

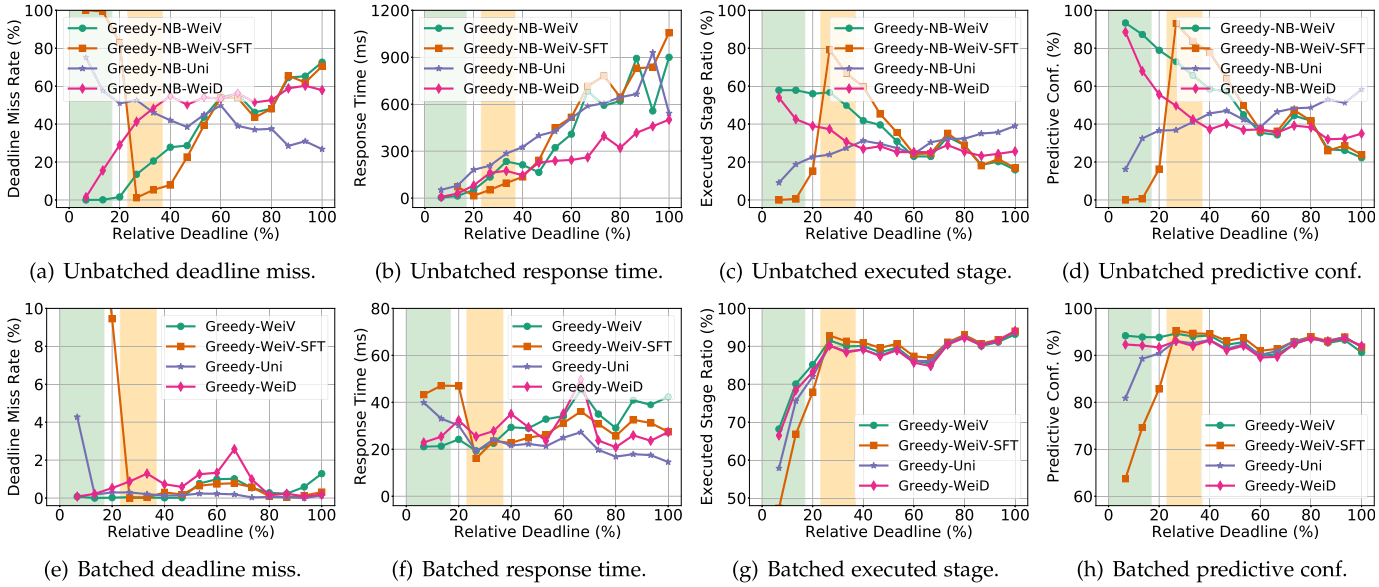
In this subsection, we implement and evaluate the distance-based criticality assignment and relative velocity-based criticality assignment, and separately compare their different versions, including a weight shift corresponding to the minimum distance l_{min} and the minimum relative deadline $\frac{d_{min}}{d_{max}}$ respectively.

In following experiments, we simulate the critical driving scenarios by setting the frame replay period to 60ms. Four metrics are compared: 1) Deadline miss rate; 2) Mean response time (i.e., the execution time of the first network stage); 3) Mean executed stage (i.e., the average ratio of executed network stage over its maximum network stage); 4) Mean predictive confidence (i.e., the mean ratio of achieved predictive confidence over the maximum confidence when all stages are executed). In the last two metrics, we compute the relative ratio to resolve the impact of specific task instances.

6.5.1 Case 1: Distance-Based Criticality

We compare the scheduling results on objects at every distance range when different weight mechanisms are applied in Fig. 6. We first look at the greedy algorithms without batching. When using uniform utility, objects are not differentiated according to their distances besides the predictive confidence, so we see close objects show much higher deadline miss rate and lower predictive confidence. Insufficient computation resources are assigned to close objects within their short deadline, who are actually more critical to the system safety. The issue is resolved in weighted greedy by partially trading the performance of distant objects. Close objects are strictly prioritized over distant objects since no task batching is applied. For close objects, significantly more computation resources are allocated, leading to the significant decrease in deadline miss, and increase in prediction quality (reflected in both executed stage and predictive confidence). The shifted version also properly adjusts the focus (i.e., prioritization) of the algorithm as expected. The prioritized distance range moves from 0-10m (highlighted in green) to 15-25m (highlighted in orange) after applying the shift. We can conclude that the weighted greedy algorithm without batching perfectly solve the problem of priority inversion, but its general performance is inferior.

Then we also compare the batched algorithms. Since batching significantly increases the task processing capacity of GPU, the deadline misses and response times of all objects are significantly reduced in the batched greedy algorithm with uniform-weight. However, there are still deadline misses on very close objects (5m) in Greedy-Uni. Our objective is to resolve deadline misses at close objects, and increase their prediction quality, considering their importance to the system safety. Misclassification at such objects can lead to serious safety issues. After applying the weight mechanism, the deadline misses on close objects are resolved, and their response times are also further reduced. More executed stages and higher predictive confidence on close objects indicate that more computation resources are allocated to them by Greedy-Wei compared to Greedy-Uni.



(a) Unbatched deadline miss. (b) Unbatched response time. (c) Unbatched executed stage. (d) Unbatched predictive conf.

(e) Batched deadline miss. (f) Batched response time. (g) Batched executed stage. (h) Batched predictive conf.

Fig. 7. Scheduling results of greedy algorithms on objects with *different relative deadlines*, when applying relative velocity-based weight (*WeiV*), distance-based weight (*WeiD*), and uniform (*Uni*) weight. We separately report the results with and without task batching. The range highlighted with green shows the advantage of *Greedy-WeiV*, and the range highlighted with orange shows the advantage of *Greedy-WeiV-SFT*.

The improvement comes at the cost of increased deadline miss and degraded prediction quality at distant objects (i.e., 45-60m). In addition, the shifted algorithm successfully ignore the closest objects (i.e., $\leq$15m) and prioritizes objects between 15-30m. Note that we set l_{min}=15m for better visualization effect.

Finally, we briefly analyze the performance of Greedy-WeiV w.r.t objects at different distance ranges. Greedy-WeiV, though being better than Greedy-Uni in prioritizing close objects because objects with short relative deadlines are mostly close, is not optimal as Greedy-WeiD because it can prioritize far objects as well if they have a fast relative velocity. In conclusion, Greedy-WeiD is the best option if we simply want to prioritize the closest objects.

6.5.2 *Case 2: Relative Velocity-Based Criticality*

Next we compare the scheduling performance when relative velocity-based criticality is applied. The associated results are presented in Fig. 7. In this experiment, we use the relative deadline $\frac{d_i}{d_{max}}$ as the x-axis, which is decided by both the object distance and its relative velocity. In the shifted weighted greedy algorithm, we set the shift $\frac{d_{min}}{d_{max}} = 20\%$. Without applying the weight mechanism, Greedy-Uni still has some deadline misses on critical objects (i.e., relative deadline $\leq$ 20 percent, as highlighted in green), even though the values are already much lower than Greedy-NB-Uni. The application of relative velocity-based criticality effectively reduces the deadline miss rate of critical objects to zero. Both response efficiency and prediction quality (i.e., executed stages and prediction confidence) on critical objects with short relative deadline are improved after applying the relative velocity-based weight mechanism. In addition, Greedy-WeiV-SFT successfully skips the objects with deadlines below d_{min}, and only prioritizes those with their deadlines larger than but close to d_{min}, as highlighted in orange. Regarding the cost, Greedy-WeiV induces slightly higher deadline miss rate and longer response times on objects with long deadlines, compared to the Greedy-Uni. The prediction qualities are quite similar between Greedy-WeiV and Greedy-Uni on objects with long relative deadlines.

We further investigate the performance of Greedy-WeiD in this experiment. It presents small deadline miss rates between 5 to 40 percent, where Greedy-WeiV shows no deadline miss. The average response time and prediction quality of Greedy-WeiD is also inferior to Greedy-WeiV in this range. Therefore, we can conclude that although Greedy-WeiD shows similar performance as Greedy-WeiV in some cases (e.g., both give no deadline miss on objects with the shortest relative deadlines), it can not replace Greedy-WeiV to provide timely and high-quality responses to all fast-approaching objects.

6.6 Breakdown Latency Quantification

In this subsection, we quantify and report the breakdown latency overhead for each step in the proposed pipeline. The frame period is set as the default 100 ms. We show the latency distribution for data slicing, and the scheduling algorithm respectively in Fig. 8. Slicing is very efficient, with an overhead below 5 ms in most cases. The average slicing latency is 2.04 ms. Object criticality computation is included in the slicing step. No noticeable latency is induced by the criticality allocation. The scheduling algorithm shows relatively higher overhead, but it is still efficient compared to the network inference time. It shows an average latency of 5.09 ms and is below 15 ms in most cases.

6.7 System Robustness Test

In this subsection, we test the system robustness to inaccuracies in bounding box location due to the approximate nature of LiDAR-based image slicing/segmentation. We test with the batched online greedy algorithm with distance-based criticality. Inaccuracy is represented by a shift ratio, R, from ground truth. Specifically, starting with the ground truth object location, we first randomly shift the bounding box center by some ratio $r \in [1, 1 + \frac{R}{2}]$, and then

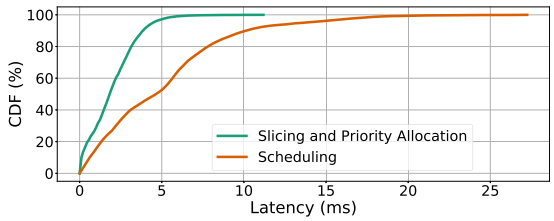
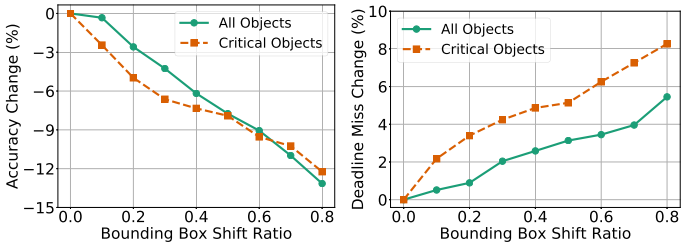


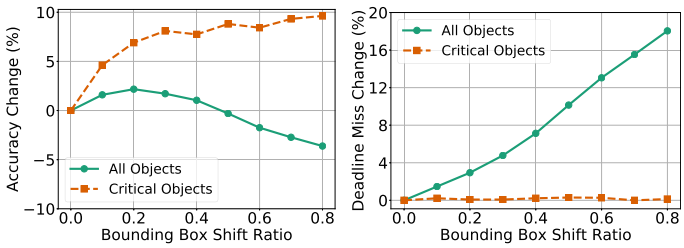
Fig. 8. Cumulative distribution of breakdown latency for each step within the proposed pipeline.



(a) Accuracy degradation. (b) Deadline miss rate change.

Fig. 9. Accuracy and deadline miss change on *ResNet*.

upscale the box size by a multiplicative factor of $r+1$ of its original size. The expansion in size reflects the intuition that when the system incorporates a less accurate (say, LiDAR-based) object localization mechanism, it makes sense to compensate by inspecting a larger area (by the camera) around the location indicated by the LiDAR. We first consider ResNet. The changes in accuracy and deadline miss rate for all objects (classified by ResNet) and for critical objects are shown in Fig. 9. Generally, the scheduling algorithm is robust against location inaccuracies, showing only about 9-13 percent degradation in accuracy and 4-8 percent increase in deadline misses for $0.6 \leq r \leq 0.8$. Critical objects show a higher increase in deadline miss rate because they are associated with tighter deadlines and larger bounding boxes. As mentioned earlier, ResNet performs classification. Thus, it cannot handle siuations where multiple objects appear in an image segment. To handle these conditions, one may use YOLO instead. Thus, we replace the classification network, ResNet, with a general object detection network (namely, YOLOv3 [42]). It can automatically localize and classify presented objects in the given image. No imprecise computations are used, since we do not have an imprecise computation model for YOLO. Accordingly, we also remove the predictive confidence term from utility, leaving only the criticality term. Results are shown in Fig. 10. An object is considered correctly classified if we have a detection that is highly overlapped with it (i.e., IoU ≥ 0.5), and has the same predicted class as the groundtruth class. On average, YOLOv3 shows a higher deadline miss rate compared to ResNet, because tasks are no longer preemptive without an imprecise computation model. However, *critical objects* suffer much fewer deadline misses with increased inaccuracy. We believe this is because the scheduling algorithm always prioritizes critical objects, since we no longer use the prediction confidence term. Interestingly, we also see that location inaccuracies (resulting in larger bounding boxes) can actually improve the prediction quality of YOLOv3. A much smaller average degradation is seen than with ResNet. Moreover, for critical objects, we observe an



(a) Accuracy degradation. (b) Deadline miss rate change.

Fig. 10. Accuracy and deadline miss change on *YOLOv3*.

improvement on accuracy, while their deadline miss rate maintains roughly stable. This is because YOLO is designed to find smaller objects in larger images. The experiment has profound implications: We do not need an accurate ranging and localization mechanism to direct the attention of the camera. An inaccurate one actually works very well (with YOLO), leading to improvements in accuracy of critical objects without degradation in schedulability.

7 Discussion

The main contribution of the proposed architecture lies in partitioning the perception input into local regions of different criticality based on a ranging sensor, and performing prioritized inference on partial data frames. Below, we first discuss the conceptual advantages and disadvatages of this research direction, followed by suggestions for future work.

7.1 To Cue or Not to Cue? That is the Question

The fundamental idea of cueing the perception module by a ranging sensor to inspect *part of the scene* first can be criticized from at least two different reliability standpoints. It is important to discuss those considerations.

First, using LiDAR (or another ranging sensor) to cue the attention of the perception module creates a dependency of the more *intelligent* neural-network on the more *primitive* ranging sensor. This dependency might seem to decrease reliability. For example, given its reduced capabilities, what if the ranging sensor misses an object? The proposed architectural change, therefore, seems to counter best-practices in building reliable systems. A counter-argument might be that we do not actually use LiDAR for detection/classification but rather as a rough cue to alert the camera to approximate regions where one might want to look for objects (based on distance anomalies). The LiDAR is good at telling distance. The camera (together with the neural network) is good at detection/classification. In fact, *ranging* is a simpler function than neural network-based classification, and so is likely to be more reliable at what it does. Thus, we use each sensor for what it's good at. The dependency is, in fact, one where a likely less reliable component (a complex neural network performing a complex cognitive function) depends on a more reliable one (simple distance measurements to guide where to look). Prior work on cyber-physical systems calls this a *well-formed dependency* [43], and argues for simplicity when higher-reliability is needed [44] (for reasons such as ease of verifiability and freedom from unintended complexity-induced bugs). It should also be possible to skew the ranging sensor towards false positives, not false

negatives. The hope is that it can eliminate at least *some* areas from consideration based on absence of low-distance measurements for those areas.

Another way the proposed architecture seems to counter best practice in reliable system design is that it makes performance optimizations that decrease accuracy (of some perception tasks) and thus, on the surface, may jeopardize safety. Jeopardizing safety to save resources is the wrong type of trade-off. A possible counter-argument is twofold: First, we view camera-based perception (and AI in general) as a *mission-critical* component, not a *safety-critical* component. A separate safety override is needed and ideally should be as simple as possible. Taking inspiration from evolution, simple human reflexes typically handle immediate threats based on simple stimuli (flinching to protect eyes from nearby projectiles, rejecting contact with very hot surfaces, coughing when objects land in the trachea, etc). Cognitive (AI-like) processing usually does higher-level planning. We make cognitive processing more efficient, but we do not touch the safety overrides, so safety should remain as is. The second observation is that the work merely *prioritizes* the processing. The real question therefore does not seem to be whether having critical stimuli handled more quickly is a good idea. Rather it seems to be whether the way we determine criticality of the stimulus is acceptable. The latter, however, is a somewhat orthogonal problem to the generalized mechanisms we develop to enforce prioritization. Hence, there may still be merit in describing those mechanisms, as well as examples of their use. With those observations, we believe that the work may promote a discussion of best design practices for real-time autonomous perception. Below, we suggest some extensions.

7.2 Criticality Assignment and Other Extensions

While the paper discussed some ways of assigning criticality, other possibilities exist that may be of interest. For example, one can generalize relative velocity to a broader notion of change (beyond the change in distance). Since the visual information on consecutive frames is highly redundant, we do not need to track everything all the time. Instead, combining distance and visual changes, we only need to be sensitive to *changes in the 3D scene* (beyond those that occur due to ego-motion). For example, when we are waiting at a traffic light, pedestrians crossing the road or flashing lights would catch more of our attention than stopped vehicles. They are not necessarily close to us or moving at a fast relative speed. Focus on areas that exhibit a bigger change essentially maximizes information flow. This would enable the system to react quickly to changes, including changes in distance that result from fast-approaching objects.

The necessity to build a tracking module, mentioned in the context of computing relative velocity (and other types of change) alludes to another possibility in computing utility weights. Namely, weights can be computed based on an information-theory-inspired metric related to information gain and surprise. This policy is inspired by the observation that if one is tracking an object, one is able to predict its future state. When the observed state of the object matches this prediction (regardless of the amount of change from the previous frames), it means that the change that occurred reflects no additional information besides what can be extrapolated from the object's past trajectory and state. In contrast, if an object appears where it was not predicted, then something unexpected has occurred that requires attention. In other words, the policy does not focus on areas of change. Rather, it focuses on areas of *unexpected* change. It is a surprise-based policy. A hybrid scheme, integrating multiple criticality policies might be another choice. How to effectively combine multiple policies in a context-driven fashion, properly adjust their composition, and resolve their conflicts is a very promising but challenging research problem. We defer this problem to future work.

Finally, there may be other ways to save resources on less-critical tasks. An example explored in a different paper is *input resizing* [45]. Less critical parts of the scene can be reduced in resolution, leading to downstream savings. It is also of interest to explore the design of criticality-based prioritization in the *absence of LiDAR*, as well as faster algorithms for distance-based clustering (for attention cueing) and use of other ranging sensors.

8 Conclusion

We investigated the advantages and limitations of a utility-based real-time task scheduling framework for machine perception pipelines to mitigate the priority inversion problem. Two core components were introduced: (a) a prioritization scheme that enables more timely response to more critical stimuli, and (ii) a batching scheme that explores the maximum parallel capacity on GPUs to improve both the response speed and inference quality. Two criticality designs (a distance-based criticality, and a relative velocity-based criticality) were proposed and implemented. Extensive evaluations on the large-scale Waymo driving dataset validates the effectiveness and efficiency of the proposed scheduling framework in removing priority inversion without sacrificing average inference quality. The sensitivity to the inaccuracies of LiDAR segmentation is a limitation of current scheduling framework, but it can be enhanced by utilizing an object detection network (e.g., YOLO). We look forward to integrating more criticality designs in our general framework in the future.

Acknowledgments

Research reported in this paper was sponsored in part by DARPA award W911NF-17-C-0099, the Army Research Laboratory under Cooperative Agreement W911NF-17-20196, NSF CNS 18-15891, NSF CNS 19-32529.

References

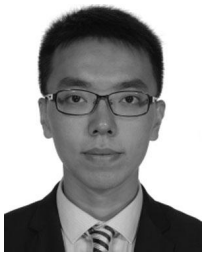
[1] S. Liu *et al.*, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 319–332.

[2] J. Sun, J. Jiang, and Y. Liu, "An introductory survey on attention mechanisms in computer vision problems," in *Proc. 6th Int. Conf. Big Data Inf. Analytics*, 2020, pp. 295–300.

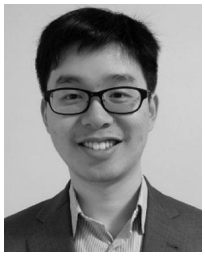
[3] M. Alcon, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Timing of autonomous driving software: Problem analysis and prospects for future solutions," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 267–280.

[4] Driverless guru, 2020. [Online]. Available: https://www.driverlessguru.com/self-driving-cars-facts-and-figures

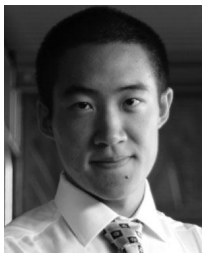
[5] D. Bamburry, "Drones: Designed for product delivery," *Des. Manage. Rev.*, vol. 26, no. 1, pp. 40–48, 2015.
[6] T. Abdelzaheretal, "Toward an internet of battlefield things: A resilience perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.
[7] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3D laser scans for online operation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 163–169.
[8] T. Abdelzaheretal, "Five challenges in cloud-enabled intelligence and control," *ACM Trans. Internet Technol.*, vol. 20, no. 1, pp. 1–19, 2020.
[9] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.
[10] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.
[11] S. Yao *et al.*, "FastDeepIoT: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proc. 16th ACM Conf. Embedded Netw. Sensor Syst.*, 2018, pp. 278–291.
[12] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 4596–4604.
[13] S. Lee and S. Nirjon, "Fast and scalable in-memory deep multitask learning via neural weight virtualization," in *Proc. 18th Int. Conf. Mobile Syst. Appl. Serv.*, 2020, pp. 175–190.
[14] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proc. 15th ACM Conf. Embedded Netw. Sensor Syst.*, 2017, Art. no. 4.
[15] B. Minnehan and A. Savakis, "Cascaded projection: End-to-end network compression and acceleration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10 715–10 724.
[16] S. Yao *et al.*, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proc. Int. Conf. Embedded Netw. Sensor Syst.*, 2020, pp. 476–488.
[17] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for GPU with preemption support," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 119–130.
[18] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "PredJoule: A timing-predictable energy optimization framework for deep neural networks," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 107–118.
[19] B. Islam and S. Nirjon, "Zygarde: Time-sensitive on-device deep inference and adaptation on intermittently-powered systems," *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 4, no. 3, pp. 1–29, 2020.
[20] M. Khayatian, M. Mehrabian, and A. Shrivastava, "RIM: Robust intersection management for connected autonomous vehicles," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 35–44.
[21] M. Bojarski *et al.*, "End-to-end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.
[22] W. Jang, H. Jeong, K. Kang, N. Dutt, and J.-C. Kim, "R-TOD: Real-time object detector with minimized end-to-end delay for autonomous driving," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 191–204.
[23] I. Baek, Z. Zhu, S. Panda, N. K. Srinivasan, S. Samii, and R. R. Rajkumar, "Error vulnerabilities and fault recovery in deep-learning frameworks for hardware accelerators," in *Proc. IEEE 26th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2020, pp. 1–10.
[24] S. Lee and S. Nirjon, "SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 15–29.
[25] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 174–187.
[26] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the NVIDIA xavier," in *Proc. 31st Euromicro Conf. Real-Time Syst.*, 2019, Art. no. 23.
[27] S. Bateni and C. Liu, "ApNet: Approximation-aware real-time neural network," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 67–79.
[28] Anyone relying on lidar is doomed, Elon Musk says, 2019. [Online]. Available: https://techcrunch.com/2019/04/22/anyone-relying-on-lidar-is-doomed-elon-musk-says
[29] Lidar vs. camera – which is the best for self-driving cars? 2020. [Online]. Available: https://medium.com/0xmachina/lidar-vs-camera-which-is-the-best-for-self-driving-cars-9335b684f8d
[30] Is Elon wrong about lidar? 2019. [Online]. Available: https://scale.com/blog/is-elon-wrong-about-lidar
[31] Billionaire austin russell's lidar firm luminar threatens tesla, 2021. [Online]. Available: https://www.flowbank.com/en/research/billionaire-austin-russells-lidar-firm-luminar-threatens-tesla
[32] J. Zhang, P. Siritanawan, Y. Yue, C. Yang, M. Wen, and D. Wang, "A two-step method for extrinsic calibration between a sparse 3D Lidar and a thermal camera," in *Proc. 15th Int. Conf. Control Autom. Robot. Vis.*, 2018, pp. 1039–1044.
[33] Tesla in Taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on, 2020. [Online]. Available: https://www.forbes.com/sites/bradtempleton/2020/06/02/tesla-in-taiwan-crashes-directly-into-overturned-truck-ignores-pedestrian-with-autopilot-on
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[35] J. W. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proc. IEEE*, vol. 82, no. 1, pp. 83–94, Jan. 1994.
[36] S. Yao *et al.*, "Eugene: Towards deep intelligence as a service," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1630–1640.
[37] S. Yao *et al.*, "RDeepSense: Reliable deep mobile computing models with uncertainty estimations," *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 1–26, 2018.
[38] S. Yao *et al.*, "Scheduling real-time deep learning services as imprecise computations," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2020, pp. 1–10.
[39] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *Proc. IEEE Int. Conf. Image Process.*, 2016, pp. 3464–3468.
[40] J. Wang, P. He, andW. Coo, "Study on the hungarian algorithm for the maximum likelihood data association problem," *J. Syst. Eng. Electron.*, vol. 18, no. 1, pp. 27–32, 2007.
[41] P. Sun *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2446–2454.
[42] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[43] H. Ding and L. Sha, "Dependency algebra: A tool for designing robust real-time systems," in *Proc. 26th IEEE Int. Real-Time Syst. Symp.*, 2005, pp. 11–pp.
[44] L. Sha, "Using simplicity to control complexity," *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, Jul./Aug. 2001.
[45] Y. Hu, S. Liu, T. Abdelzaher, M. Wigness, and P. David, "On exploring image resizing for optimizing criticality-based machine perception," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2021.



Shengzhong Liu received the BS degree in computer science from Shanghai Jiao Tong University, China, in 2017. He is currently working toward the PhD degree of computer science at the University of Illinois at Urbana-Champaign, Champaign, Illinois. His 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.



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



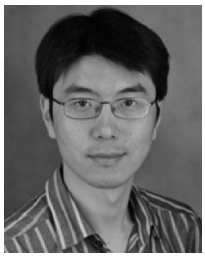
Xinzhe Fu received the BS degree in computer science from Shanghai Jiao Tong University, China. He is currently working toward the PhD degree in the Laboratory for Information and Decision Systems, and the Interdisciplinary Doctoral Program of Statistics, Massachusetts Institute of Technology, Cambridge, Massachusetts. His research interests include scheduling and optimization problems in stochastic networks.

Huajie Shao received the BS and MS degrees from Jiangnan University, China, and Zhejiang University, China, in 2011 and 2014, respectively. He is currently working toward the PhD degree of computer science at the University of Illinois at Urbana-Champaign, Champaign, Illinois. His research interests include data mining, deep learning based recommender system, social sensing. He has published more than 20 papers such as INFOCOM, Ubicomp, VLDB, Sensys, *IEEE Transactions on Signal Processing* and *IEEE Transactions on Parallel and Distributed Systems*. He received ICCPS'17 Best Paper Award and FUSION'19 Best Student Paper Award.

Rohan Tabish (Student Member, IEEE) is working toward the graduate degree in the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, Illinois. His research interests include real-time and embedded systems with a focus on the use of OS-level techniques in multi-core processors to achieve predictability and strong timing guarantees such that they can be deployed in safety-critical automotive and avionics applications. Recently, he has also been exploring Neural-Networks based software stacks to build real-time and safe AI. He is also interested in wireless communications and sensor networks.

Simon Yu received the bachelor's degree in computer engineering from the University of Illinois at Urbana-Champaign, Champaign, Illinois, in 2019. He is currently working toward the doctoral degree at the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, Illinois. His research interests include safe and fault-tolerant system design of cyber-physical systems for applications such as unmanned aerial vehicles, autonomous driving, etc.

Ayoosh Bansal received the bachelor's degree in electrical engineering from the Birla Institute of Technology and Science, Pilani, India, and the master's degree in electrical and computer engineering from the University of Wisconsin-Madison, Madison, Wisconsin. He is currently working toward the PhD degree in the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, Illinois. He is a recipient of the Saburo Muroga Endowed Fellowship. His research interests include cyber-physical systems and real-time systems, including safety, fault tolerance, security and hardware-software co-design.



Heechul Yun received the PhD degree in computer science from the University of Illinois at Urbana-Champaign, Champaign, Illinois, in 2013. He is an associate professor with the Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas. His research interests include OS, computer architecture, and real-time embedded systems with special emphasis on addressing real-time, security, and safety related issues on safety-critical cyber-physical systems (e.g., autonomous cars and UAVs). His work has appeared in top embedded real-time systems venues such as RTAS, ECRTS and Transactions on Computers; and received multiple awards and recognition (Outstanding Paper Award from RTAS'19, Best Paper Award and Outstanding Paper Award from RTAS'16, Editor's Pick of the Year Award from the *IEEE Transactions on Computers*, in 2016, Best Student Paper Nomination from RTCSA'16, and Best Paper Nomination from ECRTS'10). Prior to his PhD, he worked at Samsung Electronics as a senior software engineer.

Lui Sha (Fellow, IEEE) received the PhD degree from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1985. He worked at the Software Engineering Institute from 1986 to1998. He joined the University of Illinois at Urbana-Champaign, Champaign, Illinois, 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 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. He is a fellow of ACM.

Tarek Abdelzaher (Fellow, IEEE) received the PhD degree in computer science from the University of Michigan, Ann Arbor, Michigan, in 1999. He is currently a professor and willett faculty scholar at the Department of Computer Science, the University of Illinois at Urbana Champaign, Champaign, Illinois. He has authored/coauthored more than 300 refereed publications in real-time computing, CPS/IoT, distributed systems, intelligent networked sensing, machine learning, and control. He served as editor-in-chief of the *Journal of Real-Time Systems* for 20 years, and as associate editor of *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Embedded Systems Letters*, the *ACM Transaction on Sensor Networks*, *ACM Transactions on Internet Technology*, *ACM Transactions on Internet of Things*, and *Ad Hoc Networks Journal*. He chaired (as program or general chair) several conferences in his area including RTAS, RTSS, IPSN, Sensys, DCoSS, ICDCS, Infocom, and ICAC. His research interests include understanding and influencing performance and temporal properties of networked embedded, social, and software systems in the face of increasing complexity, distribution, and degree of interaction with an external physical environment. He is a recipient of the IEEE Outstanding Technical Achievement and Leadership Award in Real-time Systems (2012), Xerox Award for Faculty Research (2011), as well as several best paper awards. He is a fellow of ACM.