On Removing Algorithmic Priority Inversion from Mission-critical Machine Inference Pipelines

Shengzhong Liu[†], Shuochao Yao[†], Xinzhe Fu[‡], Rohan Tabish[†], Simon Yu[†], Ayoosh Bansal[†],
Heechul Yun[§], Lui Sha[†], Tarek Abdelzaher[†]
[†]University of Illinois at Urbana-Champaign, [†]Massachusetts Institute of Technology, [§]University of Kansas
Email: {sl29, syao9}@illinois.edu, xinzhe@mit.edu, {rtabish, jundayu2, ayooshb2}@illinois.edu,
heechul.yun@ku.edu, {lrs, zaher}@illinois.edu

***Abstract*—The paper discusses *algorithmic priority inversion* in mission-critical machine inference pipelines used in modern neural-network-based cyber-physical applications, and develops a scheduling solution to mitigate its effect. 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.[1] In current machine intelligence software, significant priority inversion occurs on the path from perception to decision-making, where the execution of underlying neural network algorithms does not differentiate between critical and less critical data. We describe a scheduling framework to resolve this problem, and demonstrate that it improves the system's ability to react to critical inputs, while at the same time reducing platform cost.**

***Index Terms*—Algorithmic Priority Inversion, Cyber-Physical Systems (CPS), Machine Inference.**

I. Introduction

This paper introduces the notion of *algorithmic priority inversion* that plagues modern *mission-critical* machine inference pipelines. We describe an initial solution towards removing such priority inversion from neural-network-based systems to support real-time intelligent cyber-physical applications. As a running application, we consider autonomous driving, although we expect the design principles described in this paper to remain applicable in other contexts.

Importantly, to set the scope, we distinguish between *safety-critical* and *mission-critical* design requirements of cyber-physical systems. A safety-critical requirement might be to guarantee collision-avoidance. A mission-critical requirement might be to do valid path planning, taking into account anticipated future mobility of other agents. On the surface there may appear to be overlap; the computed path must still avoid running into other objects. The distinguishing property is that the *mission-critical subsystem has lower reliability requirements*. Hence, it may speculatively use less certain data (e.g., an anticipated future trajectory of neighboring objects), and is allowed to occasionally err. The safety-critical subsystem should be able to override and restore safety when needed. For example, if the mission-critical subsystem mispredicts another object's future path, leading to a possible collision, the safety-critical subsystem should eventually detect the imminent threat and perform emergency collision-avoidance.

The application of artificial intelligence to cyber-physical systems poses different challenges in the different subsystems. One challenge is to continue to meet *safety-critical* design requirements by the safety-critical subsystem. General machine learning solutions that offer such strong safety assurances are notoriously difficult in practice and are out of scope for this paper. In fact, for the purposes of this paper, we can imagine the safety-critical subsystem to be AI-free (e.g., reliable ranging sensors that detect dangerous proximity of other objects and invoke emergency intervention).

This paper, instead, focuses on the *mission-critical* subsystem. The challenge addressed is to optimize the schedulability of mission-critical real-time perception tasks in this subsystem to remove priority inversion. Perception is one of the key components that enable system autonomy. It is also a major efficiency bottleneck that accounts for a considerable fraction of resource consumption [1], [2]. In general, priority inversion occurs in real-time systems when computations that are less critical (or that have longer deadlines) are performed together with or ahead of those that are more critical (or that have shorter deadlines). Current neural-network-based machine intelligence software suffers from a significant form of priority inversion on the path from perception to decision-making, because current algorithms process input data sequentially, as opposed to processing important parts of a scene first. This limitation may result in inferior system responsiveness to critical events, or (equivalently) increased cost of hardware to meet mission needs. By resolving this problem, we significantly improve system ability to react to critical inputs at a lower platform cost. The work applies to intelligent cyber-physical systems that perceive their environment in real time (using neural networks), such as self-driving vehicles [3], autonomous delivery drones [4], military defense systems [5], and socially-assistive robotics [6].

To understand the present gap, observe that current perception-related neural networks perform many layers of manipulation of large multidimensional matrices (called *tensors*). Yet, the current state of the art in designing the underlying neural network libraries (e.g., *TensorFlow*) is reminiscent

[1]Technically, there are two competing requirements that inform task prioritization, namely, *criticality* and *urgency*. Inversion occurs if priorities derived from these requirements are not obeyed.

of what used to be called the *cyclic executive* [7] in early operating system literature. Cyclic executives, in contrast to priority-based real-time scheduling [8], processed all pieces of incoming computation at the same priority and quality (e.g., as nested loops). Similarly, given incoming data frames (e.g., multi-color images or 3D LiDAR point clouds), modern neural network algorithms process all data rows and columns at the same priority and quality, with no regard to cues from the physical environment that impact time-constraints and criticality of different parts of the data scene.

This flat processing is in sharp contrast to the way *humans* process information. Humans have an innate ability to not only perceive their environment, but also make critical and timely attention allocation decisions that help us expend limited cognitive resources where they are most needed in a critical dynamic situation. For example, given a complex scene, such as a freeway where one of the nearby vehicles appears to have temporarily lost control of steering, human drivers are good at understanding what to focus on to plan a valid path forward amidst the resulting confusion.[2] This capability is substantially different from, say, attention mechanisms used in machine inference [9], [10], where attention is related to logical computational weights assigned to different inputs as opposed to prioritized allocation of actual processing resources.

The lack of prioritized allocation of processing resources to different parts of an input data stream (e.g., from a camera) creates what we henceforth call *algorithmic priority inversion*. In the above example, all pixels of the entire freeway scene are processed by the same algorithm at the same priority, as opposed to giving the runaway vehicle more attention while possibly temporarily ignoring other less important elements of the scene (e.g., far-away objects).

We develop an architecture for separating input data (to be processed by the neural-network) into regions of different criticality, and assigning different deadline-driven priorities to the processing of these regions. We then introduce a utility-optimizing scheduling algorithm for the resulting real-time workload to meet deadlines while maximizing a notion of global utility (to the mission). We implement the architecture on an NVIDIA Jetson AGX Xavier platform, and do a performance evaluation on the platform using real video traces collected from autonomous vehicles. The results show that the new algorithms significantly improve the average quality of machine inference, while nearly eliminating deadline misses, compared a set of state-of-the-art baselines executed on the same hardware under the same frame rate.

The rest of this paper is organized as follows. Section II introduces related work. Section III is a conceptual overview of the proposed architecture. Section IV describes the scheduling algorithms developed. An evaluation of the system using real video traces and representative autonomous driving hardware is presented in Section VI. The paper concludes with Section VII.

[2]Note that, by planning a path that continues to make forward progress, we are talking about a mission-critical function (assuming the mission involves making progress towards a destination). In contrast, a safety-critical override might simply stop the vehicle to avoid a collision. Clearly stopping the vehicle will stop progress towards mission objectives, but may ensure safety.

II. Related Work

The work is motivated by the large expansion of modern cyber-physical systems (CPS) research into areas of machine intelligence [11]–[13] and autonomy to enable progressively broader categories of tomorrow's mission-critical applications [14]. Current machine learning software has been very successful at producing run-time inference algorithms that approach or exceed capabilities of human perception [15]. Of particular promise have been recent advances in neural networks [16], [17]. 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 model compression and acceleration [18]–[20]. Examples include parameter quantization [21], edge pruning [22], node pruning [23], and dimensionality reduction (e.g., factorization [24], sparsification [25], low-rank projection [26], or domain transform [27]), as well as combinations thereof [22]. We complement that work 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. We expect that this improvement will significantly alter the price-capability trade-off of intelligent real-time embedded systems, making a new range of applications possible with increased autonomy at a lower cost.

Recent efforts on AI-empowered real-time systems addressed CPU/GPU scheduling for pipelined machine inference [28]–[34], machine-learning library optimization [35], resource and energy management [36]–[38], and communication and collaboration protocol design [39]–[45]. Several novel cyber-physical applications with deep learning were introduced [46]–[51]. Autonomous driving emerged as a flagship application motivating AI-empowered real-time system design [52]. Extensive hardware and software evaluations have been performed to understand its real time performance [1], [53]–[55]. Recent papers refactored deep neural networks to satisfy dynamic execution-time constraints during inference [35], [56]–[59]. For example, Bateni *et al.* [56] applied a combination of different layer-wise network approximation techniques to meet target deadlines. Lee *et al.* [58] introduced dynamic subnetwork construction for DNNs (where the subnetwork with best performance that meets time constraints is selected at runtime). Heo *et al.* [59] proposed multi-path neural networks for real-time object detection systems. Similarly, they dynamically change the DNN's execution path to meet deadlines. However, all these efforts are limited to configuring neural network execution for frame by frame processing. In contrast, we break-up individual frames into regions of different degree of criticality and process such regions in priority order, as opposed to the strict frame arrival (FIFO) order to mitigate algorithmic priority inversion.

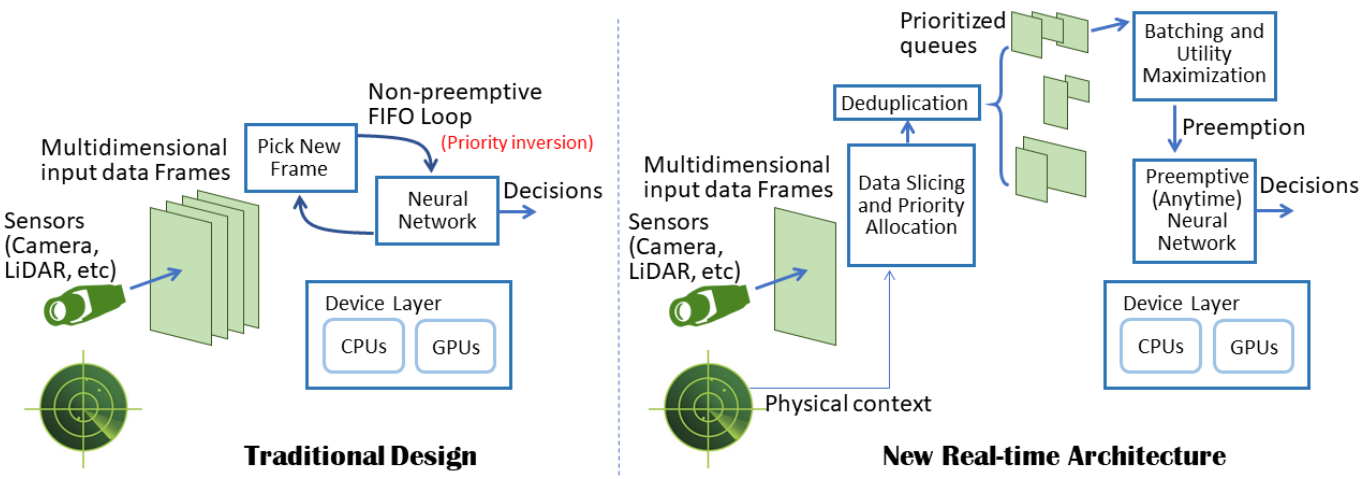


Fig. 1. Real-time Machine Inference Pipeline Architecture.

III. System Architecture

Consider an intelligent cyber-physical system 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. As mentioned earlier, we focus on scheduling of perception tasks in the mission-critical subsystem. For example, the neural network might identify the types of objects present in the field of view so that subsequent path planning can be done accordingly. Figure 1 contrasts the traditional design of machine inference pipelines in such systems to the proposed architecture. In the traditional design, input data frames captured by sensors are processed sequentially by the neural network. 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 multi-dimensional data frames captured by modern sensors (e.g., colored camera images and 3D LiDAR point clouds) carry information of different degrees of criticality in every frame.[3] Data of different degrees of criticality may require a different processing latency. For example, processing parts of the image that represent far away objects does not need to happen every frame, whereas processing nearby objects, such as a vehicle in front, needs to be done immediately because the nature of nearby objects (e.g., car versus pedestrian) has impact on immediate path planning. To accommodate these differences in input data criticality, we propose a novel mission-critical subsystem architecture that breaks the path from perception to decision-making into four components:

- *The data slicing and priority allocation module:* This module breaks up newly arriving frames into smaller regions of different degrees of criticalty based on simple heuristics (e.g., closer objects need to be attended to first).
- *The deduplication module:* This module drops redundant regions (i.e., ones that refer to the same physical objects) across successive arriving frames.
- *The "anytime" neural network:* This neural network implements an imprecise computation model that allows execution to be preempted, while yielding partial utility from the partially completed computation. The approach allows newly arriving critical data to preempt the processing of less critical data from older frames.
- *The batching and utility maximization module:* This module sits between the data slicing and deduplication modules 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). A utility maximizing algorithm controls the produced schedule to maximize a quality metric.

Since our purpose is to mitigate priority inversion on the path from *perception* to decision-making, we shall refer to the subsystem shown in Figure 1 as the *observer*. The goal is to allow the observer to respond to more urgent stimuli ahead of less urgent ones. The main contribution of this paper lies in the design of the *batching and utility maximization module* that maximizes the quality of inference while meeting response deadlines. For completeness, below we first describe all of the above components of the observer, respectively. We then detail the batching and utility maximization algorithm used.

A. Data Slicing and Priority Allocation

This module breaks up input data frames into regions that require different degrees of attention. Objects with a smaller *time-to-collision* [60] should receive attention more urgently. We further assume that the observer is equipped with a *ranging* sensor. For example, in autonomous driving systems, a LiDAR sensor measures distances between the vehicle and other objects. LiDAR point cloud based object localization techniques

[3]By different degrees of *criticality*, we are referring to different levels of importance within the *mission-critical* subsystem. For example, far-away objects are less relevant to path planning than nearby objects. We are not refering to a distinction between safety-critical and mission-critical data.

have been proposed in recent literature [61]. They provide a fast (i.e., over 200 Hz) and accurate ranging and object localization capability. The computed object locations can then be projected onto the image obtained from the camera, allowing the extraction of regions (subareas of the image) that represent these localized objects, sorted by distance from the observer. The extraction of such subareas is the main function of the data slicing module. In this paper, for simplicity, we restrict those subareas to rectangular regions. We call them *bounding boxes*. The other function of the module is prioritization (of bounding boxes) by time-to-collision, given the trajectory of the observer and the location of the object. Computing the time-to-collision is a well-studied topic and is not our contribution [60]. We list it as one of our future directions to integrate more complex and practical object priority designs into our framework.

B. Deduplication

The function of the deduplication module is very simple. It elimiates redundant bounding boxes. Since the same objects will generally persist across many LiDAR and camera frames, the same bounding boxes will be identified in multiple frames. The set of bounding boxes pertaining to the same object in different frames is called a *tubelet*. In real-time systems, in general, the best information is the most recent. Thus, only the most recent bounding box in a tubelet needs to be acted on. Boxes with significant location overlap from frame to frame are considered redundant. The deduplication module identifies boxes with large overlap and stores the most recent box only. For efficiency reasons described later, we quantize the used bounding box sizes. The deduplication module uses the same box size for the same object throughout the entire tubelet. If the underlying object changes location enough for the bounding box to jump to another size category, the overlap between the two boxes (of different size) will be small enough that the module will fail to recognize them as the same object. This creates a minor loss of deduplication efficiency but simplifies the forecasting of execution time (used by the scheduler) associated with processing what the module recognizes as the same object (since the size of its bounding box does not change).

Note that, in a traditional neural network processing pipeline, each frame is processed in its entirety before the next one arrives. Thus, no deduplication module is used. The option to add this time-saving module in our architecture arises because our pipeline can postpone processing of some objects until a later time. By that time, updated images of the same object may arrive. This enables savings by looking at the latest image only, when the neural network eventually gets around to processing the object.

C. 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 [64]–[66] 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 called to produce an *imprecise* (i.e., approximate) result.

Deep neural networks (e.g., image recognition models [62]) are a concatenation of a large number of layers that can be divided into several stages, as we show in Figure 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 [67]. Figure 3 shows the accuracy of ResNet-based classification applied to the ImageNet [63] dataset at intermediate stages of neural network processing. It shows that 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. Thus, network execution can be aborted (e.g., in favor of a new more important task) short of executing all its optional parts, yielding partial utility as described in recent work [67].

An essential component in this model is the choice of task utility, which lays the foundation for time assignment to the processing of different objects/tasks. In this paper, we set utility (from the task's output) proportionally to *confidence in result*; a low confidence output is less useful than a high confidence output. The proportionality factor itself can be set depending on task criticality, such that uncertainty in output of more critical tasks is penalized more. We adopt the algorithm proposed by Yao *et al.* in RDeepSense [68] to estimate expected confidence in outputs of future neural network stages. This allows us to compute the expected utility of each stage before it is executed.

D. 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. As discussed above, utilizing LiDAR point clouds to efficiently localize objects in each frame [61], [69], the data slicing module computes *bounding boxes* for objects detected. These boxes constitute regions that require attention, each assigned a degree of criticality. The deduplication module groups boxes related to the same object into a tubelet. Only the latest box in the tubelet is kept. The remaining boxes need not be processed. Each physical object gives rise to a separate neural network task to be scheduled. The input of that task is the bounding box for the corresponding object (cropped from the full scene). Below, we describe how the batching and utility maximization module schedules the tasks that process the different bounding boxes.

IV. The Scheduling Problem

In this section, we describe our task execution model and formulate the scheduling problem studied in this paper. We

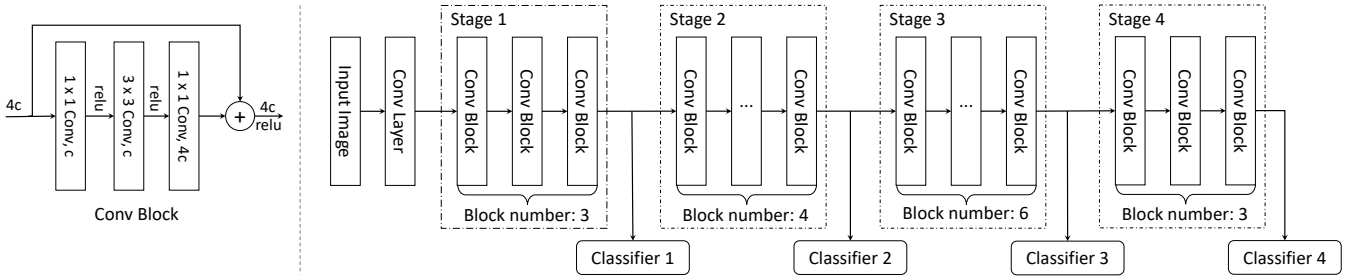
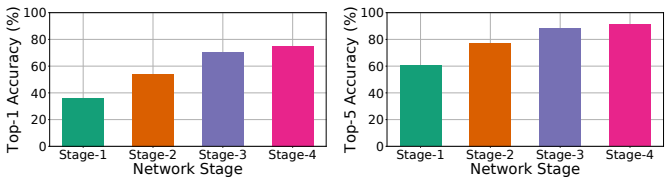


Fig. 2. ResNet [62] architecture with 4 stages and 50 layers. In the left part, we show the design of bottleneck block, which is the basic building block of ResNet. c represents the feature dimension. The classifier is simply the concatenation of a max pooling layer and a fully connected layer.



(a) Top-1 Accuracy (b) Top-5 Accuracy

Fig. 3. ResNet stage accuracy change on ImageNet [63] dataset.

then derive a near-optimal solution.

A. 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 [62], as shown in Figure 2), each processing a different input data region (a bounding box). As shown in Figure 2, tasks are broken into stages. Each stage includes multiple neural network layers. The unit of scheduling is a single stage. Neural network execution of a single stage is non-preemptive, but tasks can be preempted on stage boundaries. 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 (e.g., the corresponding object classified). The data slicing and priority assignment module is invoked at frame arrival time. Therefore, both a_i and d_i are a multiple of frame inter-arrival time, H. Since new objects can appear in the field of view at any time, 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*. A combination of two aspects make this real-time scheduling problem interesting:

1) Batching: Stages of the neural networks are executed on a GPU. We are particularly interested in lower-priced GPUs. While such GPUs feature parallel execution, 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*. The latter condition is because the processing of different bounding box sizes requires instantiating different GPU kernels. Batching that satisfies the above two conditions ensures that the same kernel is executed on all cores. Batching is advantageous because it allows us to better utilitize the GPU, so we want to take advantage of it in scheduling. To increase batching opportunities, we limit the size of possible bounding boxes used by the data slicing module to a finite set of of options. For a given bounding box size k, at most $B^{(k)}$ tasks (processing inputs) can be batched together before overloading GPU capacity. We call it the *batching limit* for the corresponding input size.

2) Imprecise Computations: Let the number of stages in the neural network for task τ_i be denoted by L_i (normally, this is the same number of all tasks, but it may depend on the size on the input object). We call the first stage *mandatory*

TABLE I
TABLE OF NOTATIONS.

Symbol	Meaning
H	Camera sampling period.
h	Time index within a period.
δ	Minimum time unit. All times are multiples of δ.
i	Object index.
j	Neural network stage index.
P	Batch of tasks.
$\mathcal{S}$	Available image size set, $\lvert\mathcal{S}\rvert = K$.
k	Image size index, which refers to the k-th image size in $\mathcal{S}$.
b	Batch size.
t	Index for scheduling period, counted in multiples of H.
τ_i	The task related to the i-th object.
s_i	Image size for the i-th object.
l_i	Number of execution stages for the i-th object.
a_i, d_i	Arrival time and deadline for the i-th object.
$e_{j,b}^{(k)}$	Exec. time of stage j when batching b images of size k.
$L^{(k)}$	Neural network stage number for image size k.
L_i	Available neural network stage number of tasks τ_i.
L	Max stage number among all image sizes.
E	Ratio between the max stage time and the min stage time.
$B^{(k)}$	Batching constraint of image size k.
B	Max batch size among all image sizes.
$R_{i,j}$	Aggregated utility for executing the i-th task for j stages.
$\mathcal{T}(t)$	Available task set at t-th scheduling period.

and call the remaining stages *optional*. Following a recently developed neural network implementation as imprecise computations [70], tasks are written such that they can return an object 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 [68]. This estimation offers a basis for assessing utility of future task stage execution. We denote the utility of task τ_i after executing $j \leq L_i$ stages by $R_{i,j}$, where $R_{i,j}$ is set proportionately to the predicted confidence in correctness at the conclusion of stage j, computed as proposed by Yao *et al.* [68]. Note that, the expected utility 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 [68].

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 j-th stage is denoted by $e_{j,b}^{(k)}$, where b is the number of tasks that are batched together during the execution of that stage. Since batched tasks execute concurrently, in principle, $e_{j,b}^{(k)}$ should not depend on b. In reality, however, there is a data copying cost in and out of the GPU that depends on the total number of batched tasks, leading to a slight increase in concurrent execution time with batching. Later in the evaluation section, we profile this effect. With the above description of our execution model, we are now ready to formulate the new scheduling problem, which we call the *Batched Online Object-recognition Scheduling with Imprecise Computation (BOOSIC)* problem, below.

B. Problem Formulation

The problem addressed in this paper is simply 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 utility, $\sum_i R_{i,l_i}$, of executed tasks is maximized, and batching constraints are met (i.e., all used GPU cores execute the same kernel at any given time, and that the batching limit is not exceeded). While the deadlines do not appear as explicit constraints in this formulation, the deadline miss ratio can be made arbitrarily small by associating deadline misses with an arbitrary large negative utility. Equivalently, one can raise the utility of timely stage execution by the same offset (and set the utility from missing a deadline to zero). In summary:

The BOOSIC problem: *With online task arrivals, the objective of the BOOSIC problem is to derive a schedule x to maximize the aggregate system utility. The schedule decides three outputs: task stage execution order on the GPU, 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 j-th 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 of the optimization problem is:*

$$BOOSIC: \max_x \sum_t \sum_i x_t(i,j)\,(R_{i,j} - R_{i,j-1})$$

$$\text{s.t. } x_t(i,j) \in \{0,1\},\ \sum_{t=1}^{T} x_t(i,j) \leq 1, \quad \forall i,j \tag{1}$$

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

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

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

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 (kernel) can be executed on the GPU at any time. However, multiple batches can be executed sequentially in one scheduling period, as long as the sum of their execution times does not exceed the period length, H.

Next, we present an online scheduling framework for reasoning about our BOOSIC problem, and propose a set of scheduling algorithms that offer different trade-off between optimality and execution overhead.

C. An Online Scheduling Framework

We derive an optimal dynamic-programming-based solution for the BOOSIC scheduling problem and express its competitive ratio relative to a clairvoyant scheduler (that has full knowledge of all future task arrivals). We then derive a more efficient greedy algorithm that approximates the dynamic programming schedule. We define the clairvoyant scheduling problem as follows:

Definition 1 (The Clairvoyant Scheduling Problem)**.** *Given information of all future tasks that will arrive, the clairvoyant scheduling problem seeks to maximize the aggregate utility obtained from (stages of) tasks that are completed before their deadlines. The maximum aggregate utility is defined as OPT.*

With no knowledge of the future, an online scheduling algorithm that achieves a competitive ratio of c (i.e., a utility greater than or equal to $\frac{1}{c} \cdot OPT$) is called c-competitive. A lower bound on the competitive ratio for online scheduling algorithms was shown to be 1.618 [71].

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, thereby allowing ourselves us to express H by an integer value (i.e., an integer multiple of the basic time unit δ). We further call the problem of scheduling current tasks within the period between successive frame arrivals, the *local scheduling problem*:

Definition 2 (The Local BOOSIC Scheduling Problem)**.** *Given the set of current tasks, $\mathcal{T}(t)$, within scheduling period, t, the local BOOSIC scheduling problem seeks to maximize the total utility gained within this scheduling period only.*

We proceed to show that an online scheduling algorithm that optimally solves the local scheduling problem within each period will have a good competitive ratio. Let L be the maximum number of stages in any task, and let B be the maximum batching size:

Theorem 1. *If during each scheduling period, the local BOOSIC scheduling problem for that period is solved optimally, then the resulting online scheduling algorithm is $\min\{2+L, 2B+1\}$-competitive (with respect to a clairvoyant algorithm).*

Proof. The proof is provided in Appendix A. □

Corollary 1. *If each task is only one stage long, and if the online scheduling algorithm solved the local BOOSIC scheduling problem in each scheduling period optimally, then the online scheduling algorithm is 3-competitive (with respect to a clairvoyant algorithm).*

Proof. This result trivially follows by substituting with $L = 1$ in the result of Theorem 1. □

D. Local Scheduling Algorithms

It remains to demonstrate how to solve the local BOOSIC scheduling problem optimally. In this section, we propose two algorithms to solve this scheduling problem. The first is a dynamic programming-based algorithm that optimally solves it but may have a higher computational overhead. The second is a greedy algorithm that is computationally efficient but may not optimally solve the problem.

1) Local Dynamic Programming Scheduling Algorithm: The resource being scheduled is the GPU. Since we only consider batching together on the GPU tasks that execute the same kernel (i.e., same stage on the same size input), we need to partition the scheduling interval, H, into sub-intervals where the above constraint is met. The challenge is to find an optimal partitioning. This question is broken into three steps:

- **Step 1:** Given an amount of time, $T_{j,k} \leq H$, what is the maximum utility attainable by scheduling the same stage, j, of tasks that process an input of size k? The answer here simply depends on the maximum number of tasks that we can batch during $T_{j,k}$ without violating the batching limit. If the time allows for more than one batch, dynamic programming is used to optimally size the batches. Let the maximum attainable utility thus found be denoted by $U^*_{j,k}$.
- **Step 2:** Given an amount of time, $T_k \leq H$, what is the maximum utility attainable by scheduling (any number of stages of) tasks that process an input of size k? Let us call this maximum utility U^*_k. Dynamic programming is used to find the best way to break interval T_k into non-overlapping intervals $T_{j,k}$, for which the total sum of utilities, $U^*_{j,k}$, is maximum.
- **Step 3:** Given the scheduling interval, H, what is the maximum utility attainable by scheduling tasks of different input sizes? Let us call this maximum utility U^*. Dynamic programming is used to find the best way to break interval H into non-overlapping intervals T_k, for which the total sum of utilities, U^*_k, is maximum.

The resulting utility, U^*, as well as the corresponding breakdown of the scheduling interval constitute the optimal solution. In essence, the solution breaks down the overall utility maximization problem into a utility maximization problem over time sub-intervals, where tasks process only a given input size. These sub-intervals are in turn broken into sub-intervals that process the same stage (and input size). The intuition why this division works is because the sub-intervals in question do not overlap. We pose an *order preserving* assumption on task marginal utilities with the same image size.

Assumption 1 (Order Preserving Assumption on Marginal Utility)**.** *For two tasks τ_{i_1} and τ_{i_2} with the same size, if for one neural network stage j, we have $R_{i_1,j} - R_{i_1,j-1} \geq R_{i_2,j} - R_{i_2,j-1}$, then it also holds $R_{i_1,j+1} - R_{i_1,j} \geq R_{i_2,j+1} - R_{i_2,j}$.*

Thus, the choice of best subset of tasks to execute remains the same regardless of which stage is considered. Below, we describe the algorithm in more detail.

Step 1: For each object size k and stage j, we can use a dynamic programming algorithm to decide the maximum number of tasks M that can execute stage j in time $T_{j,k} \leq H$. Time $T_{j,k}$ is changed between 0 and H. Observe that this computation can be done offline. The details are shown in Algorithm 1. (To simplify the notations, we ignore the object size and stage information here.) With the optimal number, M, computed for each, $T_{j,k}$, the corresponding utility, $U^*_{j,k}$, is simply the sum of utilities of the M highest-utility tasks that are ready to execute stage j on an input of size k.

Step 2: We solve this problem by a two-dimensional dynamic programming, where the two dimensions are the considered network stages and the time respectively. Given a time budget T_k, (for $0 \leq T_k \leq H$), Step 1 (above) already computed the optimal utility from assigning that time to only one stage. The recursive (induction) step takes as input the optimal utility from assigning some fraction of T_k to the first $j-1$ stages and the remainder to stage j, and computes the best possible sum of the two, for each T_k. Once all stages are considered, the result is the optimal utility, $U*_k$, from running tasks of input size k for a period T_k. The details are explained in Algorithm 2.

Step 3: Similarly to Step 2, we perform a standard dynamic programming procedure to decide the optimal time partitioning among tasks processing different input sizes. The details of this

Algorithm 1: Batching

Input: Image size index k, stage j, execution time e_b when batching b images together, batching constraint B, period H.
Output: Maximum achievable tasks $M(h)$, and optimal batch sequence $P(h)$, $\forall h \leq H$.

$M(h) = 0,\ P(h) = \emptyset,\ \forall 0 \leq h \leq H$;
for $b = 1, \ldots, B$ **do**
  if $b > M(e_b)$ **then**
    $M(e_b) := b$, $P(e_b) := \{(k, j, b)\}$;
  end
end
for $h = 2, \ldots, H$ **do**
  $h' = \arg\max_{0 \leq h' \leq h} M(h') + M(h - h')$;
  $M(h) := M(h') + M(h - h')$;
  $P(h) := P(h') \cup P(h - h')$;
end
return M, P.

Algorithm 2: Stage Assignment.

Input: Maximum tasks M, optimal batch sequence P, available task set $\mathcal{T}_j$ for each stage j, stage count L, period H.
Output: Maximum achievable utilities U_{OPT}, and optimal batch sequence P_{OPT}, $\forall h \leq H$.

$U_{OPT}(j, h) = 0,\ P_{OPT}(j, h) = \emptyset,\ \forall j, h$;
Transitted object buffer $\mathcal{T}(j, h) = \emptyset, \forall j, h$;
for $j = 1, \ldots, L$ **do**
  for $h = 1, \ldots, H$ **do**
    if $j = 1$ **then**
      $n := \min(M(j, h), |\mathcal{T}_j|)$;
      $\mathcal{T}(j, h) :=$ n tasks with max utility in $\mathcal{T}_j$;
      $U_{OPT}(j, h) :=$ total utility of $\mathcal{T}(j, h)$;
      $P_{OPT}(j, h) := P(j, h)$;
    end
    else
      $h' :=$ $\arg\max_{h' \leq h} U_{OPT}(j - 1, h') + \tilde{U}(j, h - h')$, where $\tilde{U}(j, h - h') :=$ max utility achievable with $\mathcal{T}_j \cup \mathcal{T}(j - 1, h')$ in time $h - h'$;
      $\mathcal{T}(j, h) :=$ executed tasks in $\tilde{U}(j, h - h')$;
      $U_{OPT}(j, h) := U_{OPT}(j - 1, h') + \tilde{U}(j, h - h')$;
      $P_{OPT}(j, h) := P_{OPT}(j - 1, h') \cup P(j, h)$;
    end
  end
end
return $U_{OPT}(L, h), P_{OPT}(L, h), \forall h$.

Algorithm 3: Local DP Scheduling Algorithm

Input: Available task set $\mathcal{T}^{(k)}(t)$ for each size, maximum tasks M, optimal batch sequence P, period H.
Output: Local task schedule x_t

for $k = 1, \ldots, K$ **do**
  $U^{(k)}_{OPT}, P^{(k)}_{(OPT)} :=$**Algorithm 2**$(M, P, \mathcal{T}^{(k)}(t), H)$.
end
$U_{OPT}(k, h) := U^{(1)}_{OPT}(h), \forall k, h$;
$P_{OPT}(k, h) := P_{(OPT)}(h)^{(1)}, \forall k, h$;
for $k = 2, \ldots, K$ **do**
  for $h = 1, \ldots, H$ **do**
    $h' :=$ $\arg\max_{0 \leq h' \leq h} U_{OPT}(k - 1, h') + U^{(k)}_{OPT}(h - h')$;
    $U_{OPT}(k, h) := U_{OPT}(k - 1, h') + U^{(k)}_{OPT}(h - h')$;
    $P_{OPT}(k, h) := P_{OPT}(k - 1, h') \cup P^{(k)}_{OPT}(h - h')$;
  end
end
return *The schedule* x_t *according to* $P_{OPT}(K, T)$.

Algorithm 4: Local Greedy Scheduling Algorithm

Input: Available task set $\mathcal{T}(t)$, the limitation of non-overloading batch size $B^{(k)}$ for each image index k.
Output: Local task schedule x_t

while *until the end of the period* **do**
  for $k = 1, \ldots, K$ **do**
    $\mathcal{T}_k(t) :=$ set of available tasks of size k.
    if $|\mathcal{T}_k(t)| \leq B^{(k)}$ **then**
      $U_k(t) :=$ total utility of tasks in $\mathcal{T}_k(t)$.
      $\tilde{\mathcal{T}}_k(t) := \mathcal{T}_k(t)$
    end
    else
      $\tilde{\mathcal{T}}_k(t) :=$ $B^{(k)}$ tasks with the maximum utility in $\mathcal{T}_k(t)$, $U_k(t) :=$ total utility of tasks in $\tilde{\mathcal{T}}_k(t)$.
    end
  end
  Execute the tasks in $\tilde{\mathcal{T}}_k(t)$ with the maximum value of $U_k(t)$.
end
return x_t

procedure, along with the integrated local dynamic programming scheduling algorithm is presented in Algorithm 3. With this result computed, the optimal schedule is complete.

The optimality of Algorithm 3 follows from the optimality of dynamic programming. Hence, the competitive ratio of the dynamic programming scheduling algorithm is 3 for single-stage task scheduling and $\min\{L + 2, 2B + 1\}$ for multi-stage task scheduling, according to Corollary 1 and Theorem 1, respectively. However, this algorithm may has a high computational overhead since Algorithms 2 and 3 that need to be executed each scheduling period, are $O(KLH^3)$. Next, we present a simpler local greedy algorithm, which has a better time efficiency.

2) Local Greedy Scheduling Algorithm: The greedy online scheduling algorithm solves the local BOOSIC scheduling problem following a simple greedy selection rule: execute the (eligible) batch with the maximum utility next. The pseudo-code of the greedy scheduling algorithm is shown in Algorithm 4. 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.

V. Implementation

In this section, we briefly introduce the implementation details of the proposed scheduling framework. An overview of the proposed scheduling framework implementation is demonstrated in Figure 4.

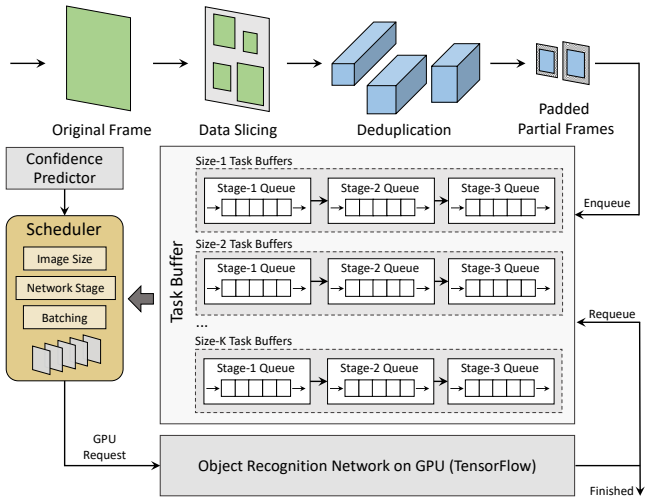


Fig. 4. System architecture for the proposed scheduling framework. All components are implemented in the user space, and the scheduled batch of images is submitted as a single GPU request.

The scheduled task is defined as the recognition of individual objects by a state-of-the-art convolutional neural network (CNN), namely the residual neural network (ResNet), which is implemented in TensorFlow [72]. To store the arrived but not finished tasks, we define a feature buffer for each (*image size, network stage*) pair. For a given image size, the buffer for each stage is intrinsically a priority queue to store the tasks waiting to execute this stage. The priority of each task is defined as its predicted marginal utility for the stage to execute. When two tasks have the same marginal utility, the one with an earlier deadline will be prioritized. When a new frame arrives, it first goes through a data slicing step, assisted by the LIDAR input, to extract the partial frames. The useless background area is removed. After filtered by the deduplication module, partial frames are padded to their closest target sizes with black borders. Finally, we push tasks into the stage-1 queues for their corresponding buffers. Similarly, when the tasks finish one stage of execution, they will be pushed into the next stage queue unless they are finished or overdue. Besides, we periodically clean up outdated tasks from each buffer to save the memory space.

Scheduling within NVIDIA GPU drivers are quite restrictive, so we follow the idea by Yao *et al.* in [70] to implement the scheduler as a middleware service in user space. It first collects the status from task buffer for each (image size, network stage) pair, which is then used as the input to our scheduling algorithm. The scheduling output tells us which image size and network stage to execute next, as well as the number of tasks to batch. Then it collects feature maps from the corresponding task buffer, and submit the batch as a single GPU request (kernel). This operation would lead to an extra memory swap between CPU and GPU on desktop machines. However, the integrated GPU of NVIDIA Jetson Xavier SoC shares the same memory with the CPU, so that the extra time delay is acceptable here.

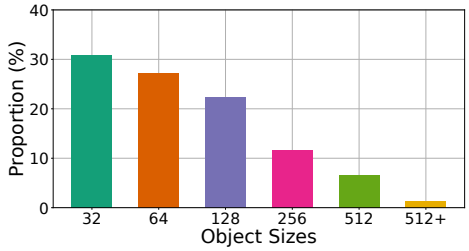


Fig. 5. Waymo object bounding box size distribution.

VI. 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.

A. Experimental Setup

1) Hardware Platform: All experiments are conducted on an NVIDIA Jetson AGX Xavier SoC, which is specifically designed for automotive platforms. It's equipped with an 8-core Carmel Arm v8.2 64-bit CPU, a 512-core Volta GPU, and 32 GB memory. Xavier delivers over 30 TOPS for deep learning applications while consuming less than 30 Watts [35]. Its mode is set as MAXN with maximum CPU/GPU/memory frequency budget, and all CPU cores are online.

2) Dataset: Our experiment is performed on the Waymo Open Dataset [73], 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. The object classes are limited to 4 classes: vehicle, pedestrian, cyclist, and sign. Only the front camera data is used in our experiment. We show the distribution of object (bounding box) sizes in Figure 5. The figure depicts the length of the longer side, rounded up to the preset bins: 32, 64, 128, 256, and 512. This also reflects the practical object size distribution from the driver's vision. Since we do not need the added resolution for identification, in our experiment, 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.

3) Neural Network Training: We use ResNet proposed by He *et al.* [62] for object classification. The network is trained on a general-purpose object detection dataset, COCO [74]. It contains 80 object classes that include those of the Waymo dataset.

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 40ms to 160ms. Since actual frame capture was done at 100ms intervals, the above corresponds to replaying the world in "slow motion" to "fast-forward" mode to

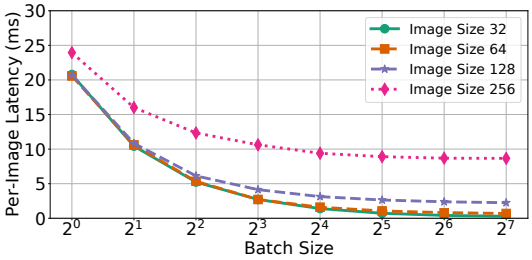


Fig. 6. Per-image latency of ResNet on NVIDIA Jetson Xavier SoC, with respect to different image sizes and batch sizes.

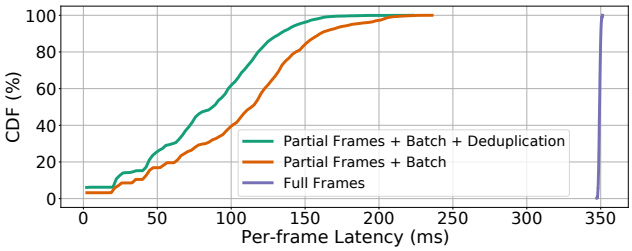


Fig. 7. Cumulative distribution of end-to-end latency on full frames with three methods. The execution time for frame slicing, deduplication (if applicable), batching, and neural network inference are all counted.

understand the impact of speed on the ability of the perception subsystem to keep up. We consider a task to miss its deadline if the scheduler fails to run the mandatory part of the task by the deadline. Otherwise, we consider the task to return a timely but possibly imprecise result. In the following evaluation, we present both the *normalized accuracy* and *deadline miss rate* for different algorithms. The normalized accuracy is defined as the ratio between achieved accuracy and the maximum accuracy when all neural network stages are finished for every object.

B. Compared Scheduling Algorithms

The following scheduling algorithms are compared.

- **OnlineDP**: the online scheduling algorithm we proposed in Section IV. The local scheduling in each period is conducted by the hierarchical dynamic programming algorithm.
- **Greedy**: the online scheduling algorithm we proposed, with the local scheduling conducted by greedy batching algorithm.
- **Greedy-NoBatch**: It always execute the object with maximal marginal utility. No batching is performed for this algorithm.
- **EDF:** It always chooses the task stage with the earliest deadline (without considering task utility).
- **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 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 as long as the deadline is not violated.
- **RR:** Round-robin scheduling algorithm. Runs one stage of each task in a round-robin fashion.

C. Neural Network Time Profiling

We first profile the inference time of ResNet on the NVIDIA Jetson Xavier SoC, for varying image sizes and batch sizes. The result is shown in Figure 6. We can observe that when the image size is small (e.g., 32×32 or 64×64), increasing the batch size is always beneficial and leads to a lower per-image latency. When the image size becomes larger (128×128 or 256×256), the benefit of parallelism gradually decreases. This is because the GPU is fully utilized at maximum parallelism, so increasing batch size increases execution time proportionally. Accordingly, we set the batching limit for each image size to be the batch size beyond which the per-image inference time stops decreasing. This size is 128, 128, 32, 8 for the four image sizes 32, 64, 128, and 256, respectively. The execution time for each valid batch is below 100ms.

D. Slicing and Batching

Next, we compare the inference time for *full frames* and *batched partial frames with/out deduplication*. In full frame processing, we directly run the neural network on image-captured full images, whose size is 1920×1280. In *batched partial frames*, we do the slicing into bounding boxes within one frame first, then perform the deduplication (if applicable), and finally batch execution of objects with 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 350 ms, while the average latency for (the sum of) batched partial frames is 105 ms without deduplication, and 83 ms with deduplication. Besides, the cumulative distributions of frame latencies for the three methods are shown in Figure 7. We can see that by batched partial frames (no deduplication), most cases have a latency below 200 ms, while deduplication further decreases the latency for most cases below 150 ms. Data slicing, batching, and deduplication steps, although induce extra processing delays, can effectively reduce the end-to-end latency. However, neither approach is fast enough compared to 100 ms sampling period, so that the imprecise computation model and prioritization are needed.

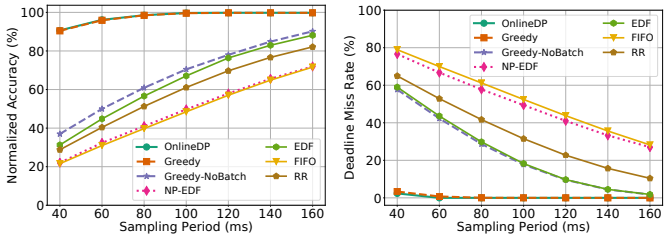
E. Scheduling Policy Comparisons

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. To isolate the effect of real-time prioritization from that of object deduplication, we turn off the latter in this part. The scheduling results are presented in Figure 8. The two proposed algorithms, OnlineDP and Greedy, clearly outperform all the baselines with a large margin in all metrics. The improvement comes for two reasons: First, the integration of the imprecise

computation model into neural networks makes the scheduler more flexible. It makes the neural network partially preemptive at the stage level, and gives the scheduler an extra degree of freedom (namely, deciding how much of each task to execute). Among stage-level scheduling algorithms, Greedy-NoBatch shows a similar deadline miss rate to EDF, but has a better accuracy. Since Greedy-NoBatch is unable to predict confidence in (i.e., utility of) future stages until it has executed one, it has no inherent way of deciding which task to start first. Thus, we select the task with the earliest deadline to execute first without violating the maximum utility rule. Second, the involvement of batching mechanism simultaneously improves the model performance and alleviate deadline misses. The batching mechanism enables the GPU to be utilized at its highest parallel capability. The deadline miss rates of both OnlineDP and Greedy are pretty close to 0 under any task load. We can also see that Greedy shows similar performance as OnlineDP, though they possess different theoretical results. One practical reason is that the utility prediction function can not perfectly predict the utility for all future stages, where the OnlineDP scheduling can be negatively impacted. Instead, Greedy only relies on the utility prediction for the next stage to make the decision.

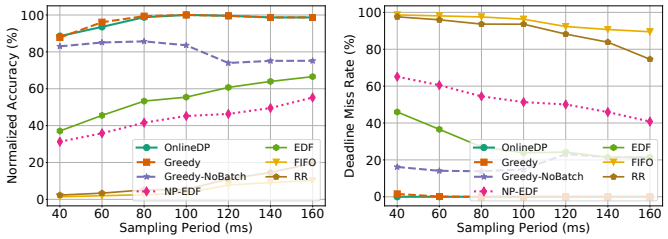
NP-EDF and FIFO have similar performance in this experiment, which might seem like a "bug". Upon closer inspection, we realize that this is because the objects in the vehicle's field of view have similar deadlines most of the time, making FIFO similar to EDF. This is expected because most of the time, when driving, no abnormal events occur that require an abrupt preemption of attention to a more critical object. While less common, such instances, however, are very important. Response to such instances is precisely what distinguishes good driving from bad driving.

To evaluate scheduling performance in driving scenarios involving the aforementioned important subcases, we compare the metrics of different algorithms for the subset of "critical objects". Critical objects are defined as objects whose time-to-collision (and hence processing deadline) fall within 1s from when they first appear in the scene. Results are shown in Figure 9. We notice that the accuracy and deadline miss rate of FIFO and RR are much worse in this case (because severe priority inversion occurs in these two algorithms). The deadline-driven algorithms (NP-EDF and EDF) can effectively resolve this issue because objects with earlier deadlines are always executed first. However, their general performance is limited for lack of utility optimization. The utility-based scheduling algorithms (Greedy, Greedy-NoBatch, and OnlineDP) are also effective in removing priority inversion, while at the same time achieving better confidence in results. These algorithms multiply a weight factor $\alpha > 1$ to increase the utility of handling critical objects, so that they are preferred by the algorithm over non-critical ones. We empirically found that $\alpha = 10$ gives good results. A more detailed exploration of this hyper-parameter will be presented in an extended report (but removed here for space limitations).



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

Fig. 8. Accuracy and deadline miss rate comparisons on all objects.



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

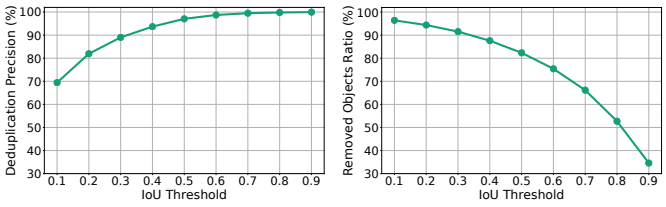
Fig. 9. Accuracy and deadline miss rate comparisons on critical objects. Critical objects are defined as objects that have a deadline less than 1s.

F. Impact of Deduplication

In this part, we report results of the deduplication module. The overlap between bounding boxes from consecutive frames is evaluated by computing their area intersection over union (IoU) score, which is defined as the ratio between their intersection area and union area. We seek to explore the best threshold for deduplication (i.e., for considering two bounding boxes to be referring to the same object). To do so, we change the IoU threshold from 0.1 to 0.9 and compute deduplication precision, define as the percentage of time the box considered to be a duplicate was indeed an image of the same object. The results are shown in Figure 10. We can precisely identify 99.5% bounding boxes belonging to the same object using a threshold of 0.7; while 99.95% of removed bounding boxes refer to the same object when the threshold is 0.9. We also report the ratio of saved workload (i.e., removed objects) in Figure 10(b). When the threshold is 0.9, we can reduce the total workload by 34.6%, while using a threshold of 0.7 can lead to 66.7% bounding boxes being removed.

G. Scheduling Algorithm Execution Time

In this part we evaluate the execution time of our proposed scheduling algorithm. Note that while the scheduled tasks run on the GPU, the scheduler runs on a single CPU core. As we mentioned in a previous statement, the OnlineDP scheduling algorithm is too slow to be applied in real time scenarios, and no computation is needed for FIFO and RR in addition to a queue. Thus, we only compare the Greedy, Greedy-NoBatch, and EDF. Specifically, we record the execution time of the scheduler on the CPU core as a percentage of the execution time of the neural networks on the GPU. Figure 11 shows the results. We can see that when the sampling period is large



(a) Deduplication precision. (b) Removed bounding box ratio.

Fig. 10. Scheduling results of Greedy algorithm after applying deduplication.

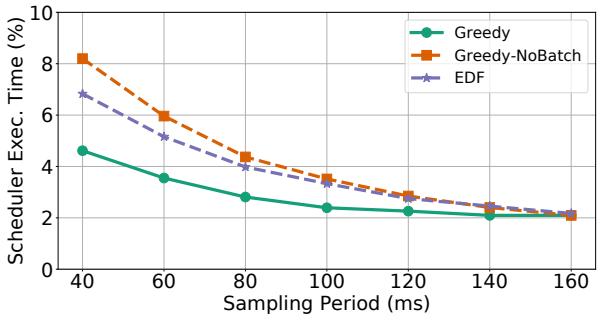


Fig. 11. Scheduling algorithm execution time comparisons.

(i.e., inverse of frame rate), all three algorithms show similar latency (around 2%), likely attributed to other CPU overheads. When the task load becomes larger, Greedy shows a better time efficiency than Greedy-NoBatch and EDF. The reason is that the batching mechanism effectively consumes more objects in the given time, so that the size of object buffer in Greedy is much smaller than Greedy-NoBatch and EDF. The execution time of our Greedy algorithm is within 5% in all evaluated workloads. Since the scheduling algorithm is executed on single CPU core, we can find that CPUs are idle most of the time. The bottleneck is indeed the GPU.

VII. Conclusions

We discussed algorithmic priority inversion in mission-critical real-time machine inference pipelines, which is found to be prevalent in conventional FIFO-based AI pipelines. To mitigate its impact, a novel online scheduling architecture was proposed with two core designs: 1) Prioritize parts of the incoming sensor data to enable more timely response to more critical stimuli; 2) Explore the maximum parallel capacity on GPU by a novel task batching algorithm to improve both the response speed and quality. We proved that through exploring optimal batching decisions within each local period, the performance of global online algorithm is also guaranteed. Extensive evaluations on a large-scale real-world driving dataset (i.e., the Waymo Open Dataset) not only validates the existence of priority inversion phenomenons, but also empirically demonstrate the effectiveness of our framework in resolving priority inversion, meeting task deadlines, and achieving better model performance. Our next steps include extending to broader scope of real-time AI pipelines (e.g., object tracking and path planning in autonomous driving), optimizing the current architecture from system implementation level (e.g., further improve the scheduler efficiency), and moving towards the safety-critical subsystems. Besides, we are also interested in applying the proposed scheduling framework to the practical autonomous driving applications.

Appendix

A. Proof of Theorem 1

We prove the theorem using charging arguments. Throughout the proof, we will refer to stages of tasks also as "tasks". We define $\{\mathcal{T}^*(t)\}_{t=1,\dots,T}$ as the set of tasks executed under an optimal Clairvoyant scheduling algorithm during each scheduling period t. Define $\{\mathcal{T}_{alg}(t)\}_{t=1,\dots,T}$ as the set of tasks executed under an arbitrary online scheduling algorithm that satisfies the condition of Theorem 1 during each scheduling period. We will charge the utility of $\{\mathcal{T}^*(t)\}_{t=1,\dots,T}$ to that of $\{\mathcal{T}_{alg}(t)\}_{t=1,\dots,T}$ using two schemes. We will show that each task in $\{\mathcal{T}_{alg}(t)\}_{t=1,\dots,T}$ is charged no more than $2+L$ times in the first scheme and no more than $2B+1$ times in the second scheme.

Consider a generic period t. In the first scheme, for each task in $\mathcal{T}^*(t)$, if its first stage is executed by the online scheduling algorithm, then we charge the utility of the task to its first stage in $\{\mathcal{T}_{alg}(t)\}_{t=1,\dots,T}$. Each task is charged for at most L times in this process. Let $\hat{\mathcal{T}}^*(t) \subseteq \mathcal{T}^*(t)$ be the tasks that are executed under the optimal Clairvoyant scheduling algorithm whose first stage are never executed under the online scheduling algorithm. Let $\hat{\mathcal{T}}_1^*(t)$ be the set of tasks in $\hat{\mathcal{T}}^*(t)$ that completely lie in period t under the schedule of the optimal Clairvoyant algorithm and let $\hat{\mathcal{T}}_2^*(t)$ be the set of tasks in $\hat{\mathcal{T}}^*(t)$ that span period t and $t+1$. By definition $\hat{\mathcal{T}}_1^*(t) \cup \hat{\mathcal{T}}_2^*(t) = \hat{\mathcal{T}}^*(t)$. Since the first stages of tasks in $\hat{\mathcal{T}}_1^*(t)$ and $\hat{\mathcal{T}}_2^*(t)$ are all available at the beginning of period t, by the concavity of the utilities of tasks, we have that the total utility of tasks in $\hat{\mathcal{T}}_1^*(t)$ and the total utility of those in $\hat{\mathcal{T}}_2^*(t)$ are both smaller than or equal to the local optimal utility. Since the considered online scheduling algorithm achieves local optimal, we can charge the utility of $\hat{\mathcal{T}}_1^*(t)$ to that of $\mathcal{T}_{alg}(t)$ with each task in the latter set being charged no more than 1 time. The same can be done for tasks in $\hat{\mathcal{T}}_2^*(t)$. Following this charging scheme for all t, we charge the utilities of all the tasks executed under the offline optimal to that of the online algorithm, with each task of the latter being charged no more than $2+L$ times.

In the second charging scheme, we use the same set of definitions. For each task in $\mathcal{T}^*(t)$, if it (the current stage) is executed by the online scheduling algorithm, then we charge the utility of the task to itself (the corresponding stage) in $\{\mathcal{T}_{alg}(t)\}_{t=1,\dots,T}$. For each set of batched tasks in $\hat{\mathcal{T}}_1^*(t)$, we select the one in the batch with the maximum utility and form a subset. Observe that, the total utility of the subset is at most OPT. Hence, the total utility of jobs in $\hat{\mathcal{T}}_1^*(t)$ is at most $B \cdot OPT$. The same can be applied to $\hat{\mathcal{T}}_2^*(t)$. Following this scheme for all t, we charge the utilities of all the tasks executed under the offline optimal, with each task executed by the online scheduling algorithm being charged no more than $2B+1$ times.

ACKNOWLEDGEMENT

Research reported in this paper was sponsored in part by DARPA award W911NF-17-C-0099, DTRA award HDTRA118-1-0026, the Army Research Laboratory under Cooperative Agreement W911NF-17-20196, NSF CNS 18-15891, NSF CNS 19-32529, and Navy N00014-17-1-2783. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies of the CCDC Army Research Laboratory, DARPA, DTRA, or the US government. The US government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] 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 *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 267–280.

[2] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

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

[4] D. Bamburry, "Drones: Designed for product delivery," *Design Management Review*, vol. 26, no. 1, pp. 40–48, 2015.

[5] T. Abdelzaher, N. Ayanian, T. Basar, S. Diggavi, J. Diesner, D. Ganesan, R. Govindan, S. Jha, T. Lepoint, B. Marlin *et al.*, "Toward an internet of battlefield things: A resilience perspective," *Computer*, vol. 51, no. 11, pp. 24–36, 2018.

[6] D. Feil-Seifer and M. J. Matarić, "Socially assistive robotics," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 24–31, 2011.

[7] T. P. Baker and A. Shaw, "The cyclic executive model and ada," *Real-Time Systems*, vol. 1, no. 1, pp. 7–25, 1989.

[8] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, 1989, pp. 166–171.

[9] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, 2015, pp. 577–585.

[10] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6077–6086.

[11] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, L. Su, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, no. 5, pp. 32–41, 2018.

[12] S. Liu, S. Yao, J. Li, D. Liu, T. Wang, H. Shao, and T. Abdelzaher, "Giobalfusion: A global attentional deep learning framework for multisensor information fusion," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–27, 2020.

[13] S. Liu, S. Yao, Y. Huang, D. Liu, H. Shao, Y. Zhao, J. Li, T. Wang, R. Wang, C. Yang *et al.*, "Handling missing sensors in topology-aware iot applications with gated graph neural network," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 3, pp. 1–31, 2020.

[14] T. Abdelzaher, Y. Hao, K. Jayarajah, A. Misra, P. Skarin, S. Yao, D. Weerakoon, and K.-E. Årzén, "Five challenges in cloud-enabled intelligence and control," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 1, pp. 1–19, 2020.

[15] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[16] 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.

[17] S. Yao, A. Piao, W. Jiang, Y. Zhao, H. Shao, S. Liu, D. Liu, J. Li, T. Wang, S. Hu *et al.*, "Stfnets: Learning sensing signals from the time-frequency perspective with short-time fourier neural networks," pp. 2192–2202, 2019.

[18] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[19] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

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

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

[22] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

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

[24] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[25] S. Bhattacharya and N. D. Lane, "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*. ACM, 2016, pp. 176–189.

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

[27] Y. Wang, C. Xu, S. You, D. Tao, and C. Xu, "Cnnpack: packing convolutional neural networks in the frequency domain," in *Advances in Neural Information Processing Systems*, 2016, pp. 253–261.

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

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

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

[31] H. Zhou, S. Bateni, and C. Liu, "S^ 3dnn: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 190–201.

[32] M. H. Santriaji and H. Hoffmann, "Merlot: Architectural support for energy-efficient real-time processing in gpus," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 214–226.

[33] M. Yang, S. Wang, J. Bakita, T. Vu, F. D. Smith, J. H. Anderson, and J.-M. Frahm, "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)*. IEEE, 2019, pp. 305–317.

[34] R. Cavicchioli, N. Capodieci, M. Solieri, and M. Bertogna, "Novel methodologies for predictable cpu-to-gpu command offloading," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[35] 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 *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, vol. 23, 2019.

[36] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.

[37] S. Bateni and C. Liu, "Predictable data-driven resource management: an implementation using autoware on autonomous platforms," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 339–352.

[38] S. Bateni, Z. Wang, Y. Zhu, Y. Hu, and C. Liu, "Co-optimizing performance and memory footprint via integrated cpu/gpu memory management, an implementation on autonomous driving platform," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 310–323.

[39] M. Khayatian, M. Mehrabian, and A. Shrivastava, "Rim: Robust intersection management for connected autonomous vehicles," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 35–44.

[40] D. Zhang, N. Vance, Y. Zhang, M. T. Rashid, and D. Wang, "Edge-batch: Towards ai-empowered optimal task batching in intelligent edge systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 366–379.

[41] J. Shin, Y. Baek, and S. H. Son, "Fundamental topology-based routing protocols for autonomous vehicles," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 265–265.

[42] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 406–418.

[43] S. Aoki and R. R. Rajkumar, "A configurable synchronous intersection protocol for self-driving vehicles," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–11.

[44] Y. Ikeda, Y. Yanagisawa, Y. Kishino, S. Mizutani, Y. Shirai, T. Suyama, K. Matsumura, and H. Noma, "Reduction of communication cost for edge-heavy sensor using divided cnn," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 244–245.

[45] S. Aoki and R. Rajkumar, "V2v-based synchronous intersection protocols for mixed traffic of human-driven and self-driving vehicles," in *2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2019, pp. 1–11.

[46] S. K. Kwon, E. Hyun, J.-H. Lee, J. Lee, and S. H. Son, "A low-complexity scheme for partially occluded pedestrian detection using lidar-radar sensor fusion," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 104–104.

[47] S. Li, D. Liu, C. Xiang, J. Liu, Y. Ling, T. Liao, and L. Liang, "Fitcnn: A cloud-assisted lightweight convolutional neural network framework for mobile devices," in *2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017, pp. 1–6.

[48] M. G. Bechtel, E. McEllhiney, M. Kim, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 11–21.

[49] K. Mikami, Y. Chen, J. Nakazawa, Y. Iida, Y. Kishimoto, and Y. Oya, "Deepcounter: Using deep learning to count garbage bags," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 1–10.

[50] M.-H. Cheng, Q. Sun, and C.-H. Tu, "An adaptive computation framework of distributed deep learning models for internet-of-things applications," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 85–91.

[51] T. Nukita, Y. Kishimoto, Y. Iida, M. Kawano, T. Yonezawa, and J. Nakazawa, "Damaged lane markings detection method with label propagation," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 203–208.

[52] M. Bojarski, "End-to-End Learning for Self-Driving Cars," *arXiv:1604*, 2016.

[53] N. Otterness, M. Yang, S. Rust, E. Park, J. H. Anderson, F. D. Smith, A. Berg, and S. Wang, "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)*. IEEE, 2017, pp. 353–364.

[54] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith, "Avoiding pitfalls when using nvidia gpus for real-time tasks in autonomous systems," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[55] J. Park, J.-H. Lee, and S. H. Son, "A survey of obstacle detection using vision sensor for autonomous vehicles," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 264–264.

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

[57] W. Kang and J. Chung, "Deeprt: predictable deep learning inference for cyber-physical systems," *Real-Time Systems*, vol. 55, no. 1, pp. 106–135, 2019.

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

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

[60] M. M. Minderhoud and P. H. Bovy, "Extended time-to-collision measures for road traffic safety assessment," *Accident Analysis & Prevention*, vol. 33, no. 1, pp. 89–97, 2001.

[61] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 163–169.

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

[63] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[64] J. W. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

[65] J. W.-S. Liu, K.-J. Lin, W. K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," in *Foundations of Real-Time Computing: Scheduling and Resource Management*. Springer, 1991, pp. 203–249.

[66] J. W. Liu, K.-J. Lin, and S. Natarajan, "Scheduling real-time, periodic jobs using imprecise results," 1987.

[67] S. Yao, Y. Hao, Y. Zhao, A. Piao, H. Shao, D. Liu, S. Liu, S. Hu, D. Weerakoon, K. Jayarajah *et al.*, "Eugene: Towards deep intelligence as a service," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1630–1640.

[68] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, pp. 1–26, 2018.

[69] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 560–565.

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

[71] B. Hajek, "On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time," in *Proceedings of the 2001 Conference on Information Sciences and Systems*, 2001.

[72] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 265–283.

[73] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

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