DNN-SAM: Split-and-Merge DNN Execution for Real-Time Object Detection

Woosung Kang[1*], Siwoo Chung[1*], Jeremy Yuhyun Kim[2], Youngmoon Lee[2], Kilho Lee[3], Jinkyu Lee[4], Kang G. Shin[5], Hoon Sung Chwa[1†]

[1]*Dept. of Electrical Engineering and Computer Science, DGIST*, Republic of Korea
[2]*Dept. of Robotics, Hanyang University*, Republic of Korea
[3]*School of AI Convergence, Soongsil University*, Republic of Korea
[4]*Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU)*, Republic of Korea
[5]*Dept. of Electrical Engineering and Computer Science, The University of Michigan – Ann Arbor*, U.S.A.
Email: chwahs@dgist.ac.kr

***Abstract*—As real-time object detection systems, such as autonomous cars, need to process input images acquired from multiple cameras, they face significant challenges in delivering accurate and timely inferences often based on machine learning (ML). To meet these challenges, we want to provide different levels of object detection accuracy and timeliness to different portions within each input image with different criticality levels. Specifically, we develop DNN-SAM, a dynamic Split-And-Merge Deep Neural Network (DNN) execution and scheduling framework, that enables seamless split-and-merge DNN execution for unmodified DNN models. Instead of processing an entire input image once in a full DNN model, DNN-SAM first splits a DNN inference task into two smaller sub-tasks—a *mandatory* sub-task dedicated for a safety-critical (cropped) portion of each image and an *optional* sub-task for processing a down-scaled image—then executes them independently, and finally merges their results into a complete inference. To achieve DNN-SAM's timely and accurate detection of objects in each image, we also develop two scheduling algorithms that prioritize sub-tasks according to their criticality levels and adaptively adjust the scale of the input image to meet the timing constraints while minimizing the response time of mandatory sub-tasks or maximizing the accuracy of optional sub-tasks. We have implemented and evaluated DNN-SAM on a representative ML framework. Our evaluation shows DNN-SAM to improve detection accuracy in the safety-critical region by 2.0–3.7× and lower average inference latency by 4.8–9.7× over existing approaches without violating any timing constraints.**

I. Introduction

Real-time object detection in autonomous vehicles (AVs) is one of the most challenging and important functions for safety. AVs are typically equipped with multiple cameras and need to be able to detect and classify objects, such as other cars and pedestrians, which are then sent to the motion planner to decide subsequent maneuvering actions. To ensure the useful and/or safe handling of detected objects, both their accurate detection and timely response (i.e., meeting the timing requirements) are key system design objectives.

There are two key characteristics to consider for real-time object detection in AVs: (1) *different criticality levels by different portions* within each scene image and (2) *dynamic variations in the safety-critical portion* by scene-by-scene. Given an input image, different portions of the image may have different levels of importance or criticality for safety. For example, other cars or pedestrians nearby on the road are more important than those in other areas because failure of their detection could lead to fatal accidents. Moreover, such a safety-critical portion of each image dynamically varies with various factors, such as the vehicle's speed/heading and moving objects in its vicinity. These distinct features must additionally provide different levels of detection quality and responsiveness to image portions of different criticality levels while adapting to dynamically-varying safety-critical portions in the driving scenes.

Recently, deep neural networks (DNNs) have been increasingly used for object detection and have made significant progress in using camera imagery to detect and classify objects. Unfortunately, the existing DNN-based object detection systems [1]–[4] are not yet directly applicable to real-time object detection while considering different image portions with different criticality levels since they use an equal amount of computation for all portions of the input image, perform computations sequentially without prioritizing safety-critical portions, and output all the results together. Moreover, most approaches cannot achieve both accurate detection and timely response on resource-constrained onboard computing platforms since the execution time and accuracy often conflict with each other. Adding sufficient DNN layers to improve detection accuracy may easily create prohibitively high computational workload, making it difficult to guarantee timely response. Although DNN compression techniques, such as quantization [5]–[8] and pruning [9]–[12], reduce the execution time to some extent, their applicability is still limited in that they generate only one network model from the original DNN, which does not dynamically adapt itself once deployed.

We develop DNN-SAM (**S**plit **A**nd **M**erge)—a dynamic split-and-merge DNN execution and scheduling framework—that leverages both image scaling and cropping, two well-known image processing techniques known to effectively reduce computational workload. Instead of processing an entire input image once in a full DNN model, DNN-SAM first splits a DNN task into two smaller sub-tasks–a primary (*mandatory*)

*Co-leading authors.
†Corresponding author.

sub-task processing a safety-critical portion only by image cropping and a secondary (*optional*) sub-task processing a down-sampled entire image by image scaling—then executes them independently, and finally merges their results into a complete one. The mandatory sub-task should be completed by the task's deadline, and the optional sub-task can be executed after the mandatory sub-task while still before the deadline using adaptive scaling if there are enough resources that are not committed to run a mandatory sub-task for any task. For each input image of a DNN task, DNN-SAM efficiently identifies its safety-critical portion, constructs corresponding mandatory and optional sub-tasks on-the-fly by the split interface, and merges their results by the merge interface, which enables seamless split-and-merge DNN execution for unmodified DNN models. DNN-SAM also provides two new scheduling algorithms—EDF-MandFirst and EDF-Slack—that determine not only the order of executing mandatory and optional sub-tasks but also the scales of optional sub-tasks at run-time by adapting to the changes in the size of safety-critical portion. EDF-MandFirst is designed to schedule mandatory sub-tasks as early as possible to detect objects in the safety-critical region as quickly as possible with a high resolution. EDF-Slack is designed to schedule optional sub-tasks with as large a scale as possible to maximize the overall accuracy while meeting all timing constraints.

DNN-SAM offers three distinct benefits. First, it is generally applicable to existing DNN-based object detection systems since it does not require any modification on existing DNN models but does only forward new inputs by scaling and cropping to DNNs. Second, it enables to provide different levels of accuracy and responsiveness to sub-tasks, i.e., faster and more accurate response in the safety-critical portion by prioritizing mandatory sub-tasks over optional ones. Third, it enables to capture dynamic variations in the size of the safety-critical portion at run-time and adaptively select the scales of optional sub-tasks so as to meet timing requirements by trading off the execution cost for accuracy.

We have implemented DNN-SAM on a representative ML framework [13] and evaluated its effectiveness in terms of detection accuracy and timely response. Our in-depth evaluation results show DNN-SAM to improve detection accuracy in the safety-critical region by 2.0–3.7× and reduce the average inference latency by 4.8–9.7× compared to existing approaches without violating any timing constraint. We have also shown, via a case study of emergency braking on a 1/10 scale AV, that DNN-SAM can improve quality of control with fast and accurate object detection, thus enhancing safety.

This paper makes the following main contributions:

- Demonstration of a case study to motivate a split-and-merge DNN execution: splitting a DNN task into two smaller sub-tasks, one for processing only a safety-critical portion and the other for processing a down-sampled image by leveraging image cropping and scaling (Sec. III);
- Development of DNN-SAM, a software framework that enables seamless split-and-merge DNN execution for unmodified DNN models (Sec. IV);
- Development of new scheduling algorithms that prioritize sub-tasks according to their criticality levels and adaptively adjust the scale of the input image to ensure the timing constraints of DNN inference tasks (Sec. V); and
- Demonstration of DNN-SAM's effectiveness, via in-depth evaluation and a case study, in terms of detection accuracy and timely response (Sec. VI).

II. Related Work

A number of recent studies have focused on timely inferences for real-time object detection. A significant proportion of existing solutions [14]–[17] focused on dynamic construction of (sub)-network in a DNN to meet the timing constraint. AnytimeNet [14] gradually inserts additional layers to support adaptive timeliness by trading off execution cost for quality of results. NestedNet [15] constructs a nested structure of a DNN to meet diverse resource requirements. Some studies [16], [17] proposed dynamic multi-path neural networks to satisfy varying timing constraints. Although all of these studies have made valuable contributions to supporting timely inferences, they have not yet been directly applicable to real-time object detection while considering different image portions with different criticality levels since they use an equal amount of computation for all portions of the input image.

Some studies [18]–[22] proposed real-time scheduling frameworks to schedule multiple DNN tasks to meet their timing constraints. ApNet [18] guarantees deadlines of DNN workloads via efficient approximation. S^3DNN [19] gathers DNN requests by data fusion and prioritizes multiple DNN tasks by supervised streaming. The work in [20] employs imprecise computation that offers mandatory and optional parts for DNNs to meet their deadlines. [21], [22] propose pipelined CPU/GPU scheduling frameworks to support concurrent execution of tasks. However, they are not able to prioritize the computation of safety-critical portions of DNN tasks over that of other portions.

A recent study [23] proposed a novel scheduling architecture that separates input data into regions of different criticality and assigns different priorities to the processing of the regions. However, they did not deal with the timing constraint when scheduling tasks, rendering them infeasible for time-critical object detection systems. DNN-SAM is differentiated from existing studies as DNN-SAM provides high accuracy while meeting timing guarantees by split-and-merge DNN execution.

III. Target System and Motivation

This section describes our target system and demonstrates, via a case study, the feasibility of leveraging image scaling and cropping for real-time object detection that motivates split-and-merge DNN execution.

A. Target System: DNN-based Object Detection

As shown in Fig. 1, an AV is typically equipped with an embedded vision system that consists of multiple cameras (e.g., front/side/rear cameras) and other sensors (e.g., LiDAR and IMU) to sense ambient environments. In such a system,

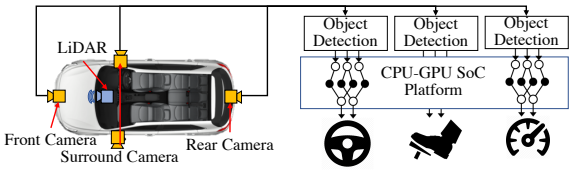


Fig. 1. An example of DNN-based object detection systems

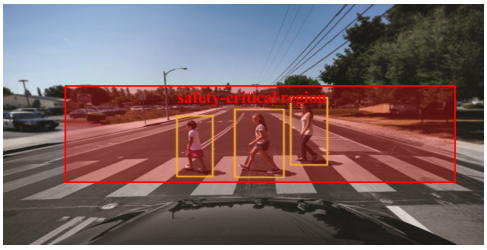


Fig. 2. A safety-critical region of the front camera

each vision task (i.e., object detection) is implemented as a *periodic task* that takes an image from a camera and performs DNN-based computation (i.e., YOLOv3 [2]) and provides useful information for motion planning, such as steering and braking, of the car; this process is repeated periodically. For instance, if the front camera detects a pedestrian, then the car generates an emergency braking signal. Therefore, vision tasks thereon must produce timely inference before their deadlines for road safety and high control quality of the car.

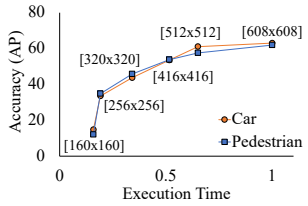
Moreover, due to the nature of autonomous driving, some portion of an image may have a different level of criticality from other portions. For instance, as shown in Fig. 2, the road where the car is heading is safety-critical since missing other cars or pedestrians in this region may cause a serious accident. Such a safety-critical portion may dynamically vary image-by-image depending on the vehicle's speed and moving objects in its vicinity, i.e., time-to-collision. Therefore, each vision task must identify a safety-critical portion of an image on-the-fly and provide faster and more accurate inference results on the safety-critical portion than non-safety-critical ones.

Such multiple DNN-based vision tasks often run on a GPU-enabled embedded board (i.e., NVIDIA Xavier system-on-chip (SoC) [24]) while sharing limited computation resources such as a single GPU instance. For safety, it is mandatory to meet the timing requirements of all vision tasks even when they share the limited resources.

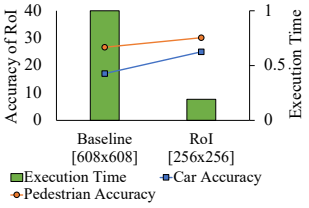
B. Feasibility of Leveraging Image Scaling and Cropping

We present a case study to investigate the effects of image scaling and cropping on the execution time and detection accuracy and check the feasibility of leveraging image scaling and cropping for real-time DNN-based object detection.

Image scaling. In order to ensure the timely response under resource constraints, we first consider adjusting computational workloads of DNN tasks through image scaling. Note that, when a smaller size of input is applied to the input layer of a DNN, the input sizes of its subsequent layers are changed accordingly. So, a DNN can alleviate its workload by processing a down-sampled image as input. Fig. 3a shows the



(a) Comparison of execution time and accuracy by down-sampling.



(b) Comparison of accuracy of RoI and execution time.

Fig. 3. Effects of image scaling and cropping on accuracy and execution time

effect of image scaling on YOLOv3.[1] In the figure, the x-axis represents the normalized execution time of YOLOv3 with different input image scales (the longest side of the image) from 160, 256, 320, 416, 512, and 608 (from the left) while the y-axis represents the corresponding accuracy in detecting cars and pedestrians. Note that the execution time is normalized to the baseline with the resolution of 608×608. We observe a trade-off between execution time and detection accuracy in choosing the input image scale. A smaller scale requires less execution time at the expense of high detection accuracy loss. As the scale increases, the execution time increases, but the accuracy loss decreases.

Image cropping. For a more responsive and accurate detection of a particular image portion, we also consider image cropping that uses the cropped image only containing the safety-critical region as input. Note that the safety-critical region will hence-forth be referred to as *Region-of-Interest* (RoI). As shown in Fig. 3b, processing a cropped image with the size of 256×256 can effectively lower the computational workload (the green bar on the right) by $5.2\times$ with no accuracy drop in the cropped portion (the orange and blue lines on the right) as compared to the original image[2]. Since the cropped region still contains an image portion with the original resolution, a DNN task can effectively reduce the execution time while preserving accuracy (or even improving accuracy as shown in the figure due to the property of YOLOv3 model in that it shows better detection performance for small objects by utilizing a fixed number of anchors).

This measurement-based case study suggests that we can *selectively* apply image scaling and cropping to different portions of an image having different criticality levels: processing a cropped image corresponding to the safety-critical region to provide high accuracy within RoI and fast response, and processing an adaptively scaled image by trading off the execution cost for accuracy to meet the timing requirements subject to available resources at run-time.

IV. DNN-SAM: System Design

A. Overview

Leveraging both image scaling and cropping explained in Sec. III-B, we design DNN-SAM, a software framework that

[1]We run YOLOv3 on an NVIDIA Xavier SoC, and inference accuracy is evaluated with 7,481 images on the KITTI in-vehicle camera dataset [25].

[2]The accuracy of RoI is defined as the average precision of detected objects inside the RoI over the total ground truth.

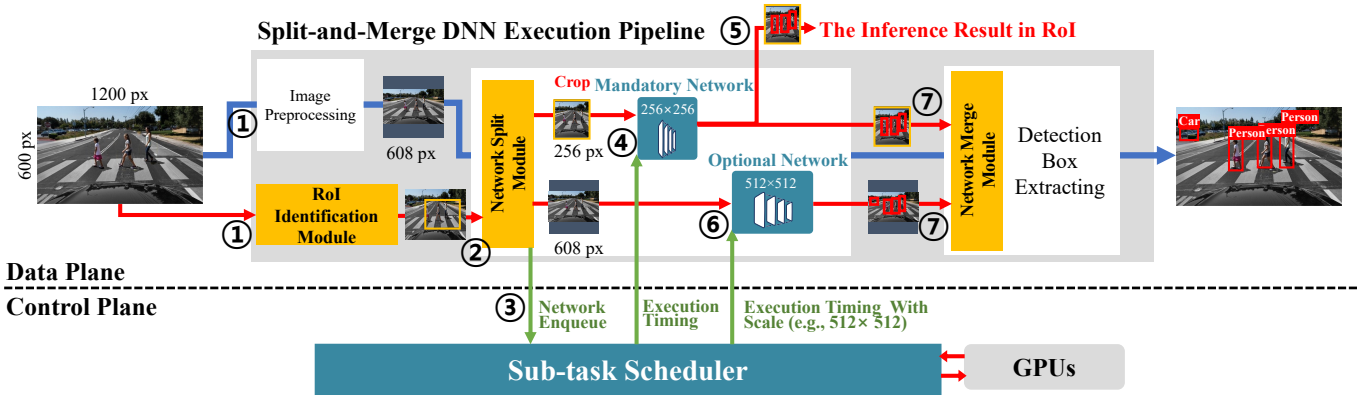


Fig. 4. System-level overview of DNN-SAM

supports split-and-merge execution for multiple DNN tasks. DNN-SAM splits a DNN task into two smaller sub-tasks—(i) a primary (*mandatory*) sub-task processing a safety-critical portion by image cropping and (ii) a secondary (*optional*) sub-task processing a down-sampled entire image by image scaling—then executes them independently, and finally merges their results into a complete one. By maximizing the image scale for optional sub-tasks without compromising the timely execution of mandatory sub-tasks, DNN-SAM can provide different levels of detection accuracy and timeliness to different portions with different criticality levels for each input image while meeting all timing constraints. To this end, the key idea behind DNN-SAM is to enable RoI identification, transparent DNN decomposition into mandatory and optional sub-tasks, prioritization of mandatory and optional sub-tasks, and adaptive scaling of optional sub-tasks. The core design features of DNN-SAM can be summarized as follows.

- **RoI identification.** DNN-SAM implements its RoI identification module using sensor fusion of camera, LiDAR, and IMU, to extract RoI within each input image at a low cost.
- **Transparent split-and-merge (S&M) DNN execution.** DNN-SAM implements split and merge operations embedded in an existing ML framework to enable seamless S&M DNN execution for unmodified DNN models.
- **Sub-task scheduling and adaptive scaling.** DNN-SAM implements a lightweight real-time scheduler that not only automatically orchestrates the execution of mandatory and optional sub-tasks at the system level but also continuously monitors available resources (slack) and adaptively selects the scales of optional sub-tasks at run-time to meet the timing requirements.

Workflow. Under DNN-SAM, as shown in Fig. 4, each image input of a DNN task is forwarded to its corresponding S&M DNN execution pipeline and RoI identification module (①). The RoI identification module segments objects and calculates the *time-to-collision* for each object using LiDAR and IMU sensors, allowing the extraction of RoI (②). The S&M DNN execution pipeline then constructs two copies of the original DNN model—one for processing a cropped image corresponding to RoI (mandatory sub-task) and the other for processing a down-scaled image (optional sub-task)—and sends a *ready* message to the scheduler (③). Then, the scheduler controls the execution times of sub-tasks (④) according to the scheduling algorithms to be discussed in Sec. V. When a mandatory sub-task completes its execution, the pipeline sends a *complete* message to the scheduler. Note that the inference result of the mandatory sub-task can be used in advance as input to other components of the car (⑤). Then, the subsequent optional sub-task becomes ready and waits for its execution by the scheduler. When an optional sub-task is scheduled for execution, the scheduler determines its scale by considering available slack (⑥) to be discussed in Sec. V. When an optional sub-task completes its execution, the pipeline merges the results of mandatory and optional sub-tasks into a complete one by removing duplicated objects detected by both sub-tasks (⑦) and sends a completion message to the scheduler.

Benefit. DNN-SAM offers three benefits. First, it is generally applicable to existing DNN-based object detection systems since it does not require any modification on the original DNN models' layer architecture and internals but does only forward new inputs by scaling and cropping to DNNs. Second, it enables a scheduler to use a smaller sub-task as the basic unit of scheduling, thus utilizing resources efficiently, and to make it possible to prioritize safety-critical mandatory sub-tasks over optional ones. Then, the outputs of mandatory sub-tasks can be used in advance as input for other computations, such as the motion planner, improving quality of control. Third, it enables adaptive selection of the scales of optional sub-tasks while considering the varying size of RoI at run-time by trading off the execution cost for detection accuracy. If there is less spare time before the deadline, an optional sub-task can select a small scale to finish its execution before the deadline at the expense of sacrificing detection accuracy. On the other hand, if there is enough spare time before the deadline, an optional sub-task can select an even larger scale than the original input image to improve detection accuracy.

B. Technical Challenges

To enable split-and-merge DNN execution, DNN-SAM addresses the following challenges:

C1. How to efficiently identify RoI?

C2. How to decompose the original DNN model into two sub-tasks in a transparent way without incurring significant

space and time overheads and enable use of the output of the mandatory sub-task in advance?

C3. How to merge the outputs of mandatory and optional sub-tasks by effectively removing duplicated objects?

C4. How to schedule sub-tasks while considering system-wide information?

C1. Since the RoI identification is included in the S&M DNN execution, its processing time should be short and predictable. One may consider a DNN-based approach to extract RoI, but it usually incurs a high computation, making timely response of S&M DNN execution difficult. Instead, DNN-SAM employs a *sensor fusion* approach with camera, LiDAR, and IMU sensors that is known to be able to localize objects fast and accurately, and thus is widely used in AVs [26].

C2. DNN-SAM aims to provide seamless S&M DNN execution for unmodified DNN models. Current state-of-the-art ML frameworks employ a single process per DNN model [22]. One simple approach to split a DNN task is to fork two independent processes for mandatory and optional sub-tasks, but it creates 2x the memory footprint, thus making it infeasible for resource-constrained computing platforms. Another simple approach is to use a single process with a single thread which executes mandatory and optional sub-tasks sequentially. However, this cannot produce the output of a mandatory sub-task in advance without modifying the user-level source code. Instead, DNN-SAM supports a *multi-thread execution* model that enables seamless S&M DNN execution for unmodified DNN models with minimal memory overhead and early use of the output of the mandatory sub-task.

C3. In order to merge the outputs of mandatory and optional sub-tasks, the objects detected by mandatory and optional sub-tasks should be localized in a global coordinate system, and duplicated objects detected by both mandatory and optional sub-tasks should be removed for better accuracy. Intersection over union (IoU) is the most widely used evaluation metric to measure the area of overlap between two bounding boxes. Although IoU is powerful and efficient, it is not enough for S&M DNN execution. Since a mandatory sub-task performs an inference job with a cropped image, some objects at the edge are detected cut off. Such cut-off objects result in low IoU scores with their corresponding intact objects detected in a scaled image by optional sub-tasks. DNN-SAM supports a coordinate transformation formula to localize objects in a global coordinate system and a new evaluation metric that can handle those cut-off objects in addition to IoU.

C4. State-of-the-art ML frameworks employ a simple scheduling principle, i.e., FIFO, that sequentially schedules DNN tasks without prioritizing them according to their timing requirements, i.e., without considering real-time DNN tasks with different timing requirements. Moreover, the ML frameworks cannot consider system-wide information since they work as a library. Instead, DNN-SAM supports a lightweight real-time scheduler to automatically orchestrate the execution of mandatory and optional sub-tasks at the system level and to adaptively select the scales of optional sub-tasks at run-time subject to available slack to meet the timing requirements.

C. RoI Identification Module

The *RoI identification* module extracts RoI from the input image using the following three steps: i) LiDAR segmentation, ii) bounding box projection, and iii) time-to-collision (ToC) calculation. The LiDAR segmentation step receives 3D range data from a LiDAR sensor as input and segments the data into individual objects by employing a range image-based segmentation technique [27], which enables fast and accurate localization of objects. Specifically, the range data, which is out of the Field-of-View of a camera, is filtered out to perform object segmentation and distance estimation within each image input of a camera. The bounding box projection step receives the 3D position information (bounding boxes) from detected objects as input and projects each bounding box onto a corresponding 2D camera input image with the distance information by employing the technique in [28]. Step iii computes the ToC for each object based on its distance from the vehicle and the vehicle's velocity measured by IMU sensors [29].[3] Finally, the location and size of a RoI are determined so that all objects within a constant ToC threshold are included. For simplicity, we restrict RoI to a rectangular region and set a constant ToC threshold to 2 seconds, i.e., 33m away from the vehicle traveling at $60km/h$.

D. Network Split Module

To enable the *multi-thread execution* model, the *network split* module decomposes a DNN task into two sub-tasks by creating two inference threads (i.e., a thread for a mandatory sub-task and that for an optional sub-task) that share the same context of their DNN task. Since all threads in a process use identical address space, the inference threads inherently share all their network model parameters, e.g., inputs, weights, and outputs. One concern is a possible race condition when the two threads access the network parameters at the same time, but, during S&M DNN execution, both threads access the network parameters in a mutually exclusive manner as they are executed sequentially. Another issue is that an optional thread may overwrite the inference result of a mandatory one. To avoid this, DNN-SAM provides a dedicated memory space for each thread to store its inference result while sharing other network parameters. Besides, the multi-thread execution model allows the mandatory thread to immediately return its inference result even before completing the whole S&M pipeline without any modification of the user-level source code.

To address the transparency issue, we focus on fully convolutional networks (FCN) [31] that naturally operate on an input of *any* size and produce an output of the corresponding (possibly re-sampled) spatial dimensions. Thanks to such a powerful feature of FCNs, DNN-SAM does not require any modification on the original DNN model's layer architecture and internals. Instead, different inputs are fed into the original DNN model by two threads, i.e., cropped and scaled images for mandatory and optional threads, respectively. Then, each

[3]Note that estimating safe time-to-collision is a topic of active research [30], but it is beyond the scope of this paper.

thread goes through a forward pass of the DNN model with its input and produces an output whose size is determined corresponding to its input size. Due to such a powerful feature, FCN-based object detection systems are now dominant in the field [32]. Thereby, DNN-SAM can transparently provide S&M DNN execution for various FCN-based object detection systems. Note that some techniques have been proposed to efficiently convert a non-FCN network to an equivalent FCN network [31].

E. Network Merge Module

The *network merge* module localizes the detected objects in a global coordinate system and eliminates duplicated objects, if any. First, each object detected by a mandatory sub-task should be projected onto the original input space, in order to merge the result with that of the corresponding optional sub-task. In particular, as the inference result of a mandatory sub-task, the location and size of each object's bounding box, denoted as $\{(x_a, y_a), (w_a, h_a)\}$, are represented in the RoI space, where (x_a, y_a) and (w_a, h_a) are the center position and size of the box in the RoI space, respectively. Note that the location and size of RoI are represented as $\{(x, y), (w, h)\}$ in the (global) input space. Then, the location and size of each object's bounding box in the RoI space can be transformed to those in the global space, denoted as $\{(x'_a, y'_a), (w'_a, h'_a)\}$, and they are calculated as $x'_a = x_a + (x - w/2)$, $y'_a = y_a + (y - h/2)$, $w'_a = w_a$, and $h'_a = h_a$. Note that $(x - w/2, y - h/2)$ is the upper-left coordinates of RoI in the input space. In addition, as the inference result of an optional sub-task, the location and size of each object's bounding box in the scaled space, denoted as $\{(x_b, y_b), (w_b, h_b)\}$, can be transformed to those in the global space as $\{(x'_b, y'_b) = \alpha \cdot (x_b, y_b), (w'_b, h'_b) = \alpha \cdot (w_b, h_b)\}$, where α denotes the scaling factor of the optional sub-task.

With those objects localized in a global coordinate system, the merge module then performs deduplication through *intersection over union* (IoU) and a new redundancy detection metric. A typical way of detecting duplications is using IoU that can be simply computed as area of intersection divided over area of union. If the IoU score between two bounding boxes is greater than a threshold, one of the boxes is eliminated. However, our S&M DNN execution may generate a cut-off object that may have a very low IoU score with its corresponding intact object detected in a scaled image due to a small area of intersection between the two objects, failing to detect duplication. To solve this problem, we introduce a new evaluation metric for handling the cut-off objects. Instead of dividing by area of union, the proposed metric divides area of intersection by area of cut-off object's bounding box. If the newly defined score between two bounding boxes of a cut-off object in a cropped image and an intact object in a scaled image is greater than a threshold, one of the boxes is eliminated. In this way, the network merge module can merge the inference results of mandatory and optional sub-tasks into a complete one by effectively eliminating redundant objects detected by both sub-tasks, achieving better accuracy.

F. Sub-task Scheduler

To orchestrate the execution of sub-tasks, DNN-SAM supports a *sub-task scheduler* which runs as a background daemon to communicate with S&M DNN execution pipelines via a *Named Pipe IPC*, which facilitates an efficient communication between sub-tasks and the scheduler. The scheduler maintains each DNN task's S&M DNN execution model parameters (in Sec. V-A) and a priority queue for scheduling mandatory and optional sub-tasks according to their priorities. When a sub-task is ready to execute, it sends a *ready* message to the scheduler, which then enqueues the sub-task into the priority queue with its PID and deadline information. When a sub-task finishes its execution, it sends a *complete* message to the scheduler, which then dequeues another sub-task from the queue according to the scheduling algorithms (in Sec. V-B). Also, the scheduler continuously calculates available slack at each invocation (either task release or sub-task completion) and determines the scales of optional sub-tasks ready to execute. We assume the non-preemptive execution of each sub-task due to the non-preemptive feature of GPU processing.

V. Scheduling of Multiple DNN Tasks for Object Detection

While Sec. IV explained the system architecture for DNN-SAM that addresses C1–C3, it did not present how to schedule sub-tasks of multiple DNN tasks (i.e., C4). In this section, we address the following issues regarding C4, which enables DNN-SAM to maximize the overall detection accuracy while meeting all timing constraints: (a) how to model each DNN task associated with the proposed S&M DNN execution? and (b) how to schedule sub-tasks of multiple DNN tasks and how to provide offline timing guarantees for the scheduling?

A. Split-and-Merge DNN Execution Model

The original DNN inference tasks are represented as the periodic task model [33] which has been used in most real-time systems. We assume each task uses one DNN model and makes one inference request per job (task's instance). Basically, we tailor the imprecise computation model [34] to DNN-based object detection tasks. Under the S&M DNN execution model, each DNN task $\tau_i \in \tau$ consists of a mandatory sub-task τ_i^M and an optional sub-task τ_i^O, and it can be specified as $\tau_i = (\tau_i^M, \tau_i^O, T_i, D_i)$, where T_i is the period and D_i is the relative deadline equal to T_i.

Important parameters to describe the execution behavior of τ_i are τ_i^M and τ_i^O. A mandatory sub-task τ_i^M processes a cropped RoI image, and is further specified as $\tau_i^M = (R_i^M, C_i^M)$, where $R_i^M = \{(x, y), (w, h)\}$ is specified by the center position (x, y) and the *maximum* size (w, h) of RoI, and C_i^M is the worst-case execution time of τ_i^M to process R_i^M. Note that the position and size of RoI vary with an input image for each job instance to be determined at run-time, while the maximum size of RoI and the worst-case execution time C_i^M are assumed known *a priori*, e.g., in our case $w = h = 256$, according to the driving environment, such as camera's field-of-view (FOV) and the vehicle speed limit. We calculate the

worst-case execution time C_i^M by decomposing it into the three components:

$$C_i^M = c_i^{\texttt{RoI}} + c_i^{\texttt{Split}} + c_i^{\texttt{Infer}} \quad (1)$$

where $c_i^{\texttt{RoI}}$, $c_i^{\texttt{Split}}$, and $c_i^{\texttt{Infer}}$ denote the worst-case execution times of RoI identification, network splitting, and inference for R_i^M, respectively, to be measured in Sec. VI-A. An optional sub-task τ_i^O processes a down-sampled entire image, and is further specified as $\tau_i^O = (S_i^O, C_i^O)$, where S_i^O is a finite set of scales (pixels of the longest side), e.g., in our case $S_i^O = \{0, 160, 256, 320, 416, 512, 608, 672\}$, and C_i^O is a set of the worst-case execution times at different scales, i.e., $C_i^O = \{C_i^O(s_i) | s_i \in S_i^O\}$. Note that an optional sub-task combines the detected objects from a down-sampled image with those from a cropped RoI by a mandatory sub-task to increase overall accuracy. We calculate the worst-case execution time at scale s_i (denoted as $C_i^O(s_i)$) by decomposing it into the two components:

$$C_i^O(s_i) = c_i^{\texttt{Infer}}(s_i) + c_i^{\texttt{Merge}}, \quad (2)$$

where $c_i^{\texttt{Infer}}(s_i)$ is the worst-case inference time for an input image of scale s_i, and $c_i^{\texttt{Merge}}$ is the maximum time to merge the inference results of mandatory and optional sub-tasks, each of which will be measured in Sec. VI-A. The period and deadline of the sub-tasks τ_i^M and τ_i^O are the same as those of task τ_i.

Each task τ_i is assumed to generate potentially an infinite sequence of jobs every T_i time-units. For each job, the execution of its mandatory sub-job must be completed within a relative deadline of D_i time-units, and its optional sub-job becomes ready for execution only when the mandatory sub-job completes. While the parameters in τ_i^M and τ_i^O are known *a priori*, the actual position and size of RoI and its corresponding execution time ($\leq C_i^M$) for a mandatory sub-job is determined by the RoI identification module in DNN-SAM at run-time. In addition, the actual scale of an input image for an optional sub-job is determined at run-time, and it may vary from instance to instance by accounting for the available resources before its deadline (to be discussed in Sec. V-B).[4] When it comes to preemptiveness, the executions are non-preemptive in a sub-task level; this means, it is impossible for any higher-priority sub-task τ_j^M (or τ_j^O) to preempt a currently-executing sub-task τ_k^M (or τ_k^O), while a sub-task τ_i^M (or τ_i^O) can be executed between the execution of the mandatory sub-task τ_k^M and that of the corresponding optional sub-task of τ_k^O.

Note that the S&M DNN execution model is more general than the original DNN model that processes an entire image once. In fact, a DNN task composed of a mandatory sub-task with an entire image as RoI and an optional sub-tasks with $S_i^O = \{0\}$ is equivalent to the original model.

[4]In periodic task systems, at most one active job per task exists in any time slot, and hence, for simplicity of presentation, we use the term task (sub-task) also refers to active job (sub-job) of a task in the rest of this paper.

B. Scheduling of Multiple DNN tasks

Building upon DNN-SAM, we develop two new scheduling algorithms that determine not only the order of executing mandatory and optional sub-tasks but also the scale of optional sub-tasks for each job. Associated with the S&M DNN execution model, we have the following scheduling objectives that must be achieved: (O1) scheduling mandatory sub-tasks as early as possible to detect objects in RoI as quickly as possible and (O2) scheduling optional sub-tasks with a large scale to maximize the overall accuracy, while meeting the deadlines of all the jobs of mandatory *and* optional sub-tasks. Both of these two objectives are important, yet often incompatible with each other.

The main challenge in achieving these scheduling objectives arises from the fact that the execution time of a mandatory sub-task varies with the size of RoI within its image input. Without considering such dynamic execution behavior that depends on the RoI size, we may severely under-utilize computing resources or substantially degrade the accuracy of optional sub-tasks. Suppose every mandatory sub-task τ_i^M is always assigned its resource based on the maximum size of RoI and its corresponding worst-case execution time C_i^M. Then, a considerable amount of computing resources will remain unused throughout all the periods where the actual size of RoI is smaller than the maximum. If those unused resources could be reclaimed and utilized for optional sub-tasks so as execute them with larger scales, we can improve the overall accuracy while guaranteeing the deadlines of all sub-tasks. We, therefore, develop online scheduling algorithms that reclaim unused resources efficiently under the varying size of RoI, and adaptively select the scales of optional sub-tasks at run-time so as to maximize the overall accuracy without compromising schedulability.

In particular, we propose two different scheduling policies: i) EDF-MandFirst and ii) EDF-Slack. For both scheduling policies, the scheduler is invoked upon release of a new job, or completion of either job's mandatory or optional sub-task. Upon each invocation, active sub-tasks in the ready queue are scheduled under the corresponding scheduling policy. EDF-MandFirst and EDF-Slack employ not only different prioritization schemes for mandatory and optional sub-tasks, but also different mechanisms of determining the scales of optional sub-tasks by utilizing available slack resources at run-time.

Determining the priority ordering. EDF-MandFirst assigns *statically* higher priorities to mandatory sub-tasks over optional sub-tasks. Optional sub-tasks are scheduled whenever no mandatory sub-task is ready in the system. We implement this by maintaining two queues: one for active mandatory sub-tasks and the other for active optional sub-tasks. The priority ordering of mandatory sub-tasks is determined according to the Earliest Deadline First (EDF) policy [33], and the same applies to optional sub-tasks. EDF-MandFirst is designed to achieve the first objective O1 by securing mandatory sub-tasks from the potential interference of optional sub-tasks. On the other hand, EDF-Slack assigns priorities to sub-tasks by the EDF

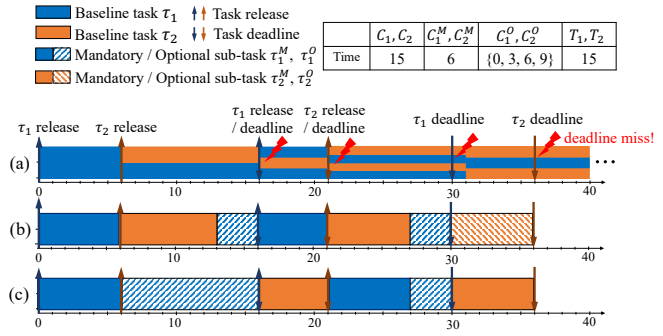


Fig. 5. Execution timeline of multiple DNN tasks under (a) Baseline, (b) EDF-MandFirst, and (c) EDF-Slack scheduling policies.

policy without separation between mandatory and optional sub-tasks. That is, optional sub-tasks with earlier deadlines are assigned higher priorities than mandatory sub-tasks with later deadlines. EDF-Slack is designed to achieve the second objective O2 by securing more resources for optional sub-tasks as long as all mandatory sub-tasks finish their execution before their deadlines. To achieve O1 and O2 *together*, EDF-MandFirst and EDF-Slack need their own mechanisms to determine the scale of the optional sub-task selected to be scheduled.

To compare the proposed scheduling policies and the baseline, let us consider a task set with the following two tasks: $\tau_1 = \tau_2 = (C_1^M = 6, C_1^O = \{0, 3, 6, 9\})$. The original DNN model of τ_1 and τ_2 has the worst-case execution time of 15. Consider the scheduling scenario shown in Fig. 5. We compare (a) Baseline (vanilla DarkNet), (b) EDF-MandFirst, and (c) EDF-Slack. In the case of Baseline shown in Fig. 5a, the response time of a higher-priority job becomes unpredictably long and misses its deadline. Without real-time scheduling support, the execution of layers of different tasks can be interleaved, since each DNN task consists of a sequence of multiple layers with data dependency. Then, a lower-priority job (e.g., the first job of τ_2) imposes interference on a higher-priority job (e.g., the first job of τ_1), missing the deadline of the higher-priority job. In the case of EDF-MandFirst shown in Fig. 5b, τ_2's mandatory sub-task can run as soon as τ_2 is released at time 6 and 21 by prioritizing mandatory sub-tasks over optional ones, and thus producing the most responsive inference for RoI. In the case of EDF-Slack shown in Fig. 5c, when the first job of τ_2 is released at time 6, τ_1's optional sub-task can still run for 6 time-units with a larger scale until the slack depletes. When the second job of τ_2 is released at time unit 21, τ_1's optional sub-task can run only for 3 time-units with a smaller scale to avoid missing the deadline of τ_2 at time 36. This way, EDF-Slack's adaptive image scaling achieves the most accurate inference while meeting the deadlines of all mandatory sub-tasks.

Determining the scale of optional sub-tasks for EDF-MandFirst. Now, let us explain how EDF-MandFirst determines the scale of optional sub-tasks. When an optional sub-task τ_k^O is selected to be scheduled by EDF-MandFirst at time t_{cur}, we need to determine the scale of τ_k^O to achieve O1 and O2. To this end, we use the following two properties. First, from the priority ordering of EDF-MandFirst, there is no active mandatory sub-task at t_{cur}. Second, since each task invokes a series of jobs periodically as mentioned in Sec. V-A, no mandatory job is released in $[t_{cur},\ d_1(t_{cur}))$, where $d_1(t_{cur})$ is the earliest absolute deadline after t_{cur} among all mandatory sub-tasks. Using the two properties, we can guarantee that there is no execution of any mandatory sub-task $[t_{cur},\ d_1(t_{cur}))$, and therefore the optional sub-task τ_k^O can utilize $d_1(t_{cur}) - t_{cur}$ amount of slack time without delaying the execution of any mandatory sub-task. EDF-MandFirst maximally uses the slack time, by determining the scale of the optional sub-task to be selected at t_{cur}, as $\arg\max_{s_k \in S_k^O} \{C_k^O(s_k) | C_k^O(s_k) \leq d_1(t_{cur}) - t_{cur}\}$.

From the policies of determining the priority ordering and the scale of optional sub-tasks, we can directly apply a non-preemptive EDF schedulability analysis [35] to EDF-MandFirst, as presented in the following theorem.

Theorem 1: If Eq. (3) holds, τ is schedulable by EDF-MandFirst (i.e., it guarantees no sub-job deadline miss).

$$\frac{max_{\tau_i^M \in \tau} C_i^M}{min_{\tau_i^M \in \tau} T_i} + \sum_{\tau_i^M \in \tau} \frac{C_i^M}{T_i} \leq 1.0 \tag{3}$$

Proof: Due to the mechanism of determining the scale of optional sub-tasks for EDF-MandFirst, the following two properties holds. First, the execution of each optional sub-task does not delay that of any mandatory sub-task (because of each optional sub-task's execution finishes before $d_1(t_{cur})$). Second, each optional sub-task does not miss its deadline (because its execution time is less than $d_1(t_{cur}) - t_{cur}$ and its deadline no earlier than $d_1(t_{cur})$). Therefore, the proof is equivalent to prove $\{\tau_i^M\}$ (without any execution of $\{\tau_i^O\}$) is schedulable by the vanilla non-preemptive EDF scheduling.

Then, the theorem holds because Eq. (3) is a simpler version of Theorem 2 in [35]. Here is a high-level idea why Eq. (3) is a sufficient schedulability condition. While the EDF utilization bound for a set of preemptive tasks is equal to 1.0 [33], that for a set of non-preemptive tasks is decreased due to the blocking of non-preemptive executions [35]. The execution of a higher-priority task is blocked by at most one currently-running lower-priority task at the release time of the higher-priority task. In the worst-case scenario, a task might be blocked by the task which has the maximum execution time. The maximum blocking factor is contributed by the task which has the shortest period. ■

Determining the scale of optional sub-tasks for EDF-Slack. Different from EDF-MandFirst, EDF-Slack selects an optional sub-task τ_k^O to be scheduled at t_{cur} even though there is a mandatory sub-task which is active at t_{cur} but has a later deadline than that of τ_k^O. We would like to determine the scale of the optional sub-task τ_i^O to be scheduled for EDF-Slack such that (i) the execution of τ_k^O does not compromise the deadline guarantee of every mandatory sub-task, and (ii) EDF-Slack can use the same schedulability analysis as Theorem 1. For (ii), we consider the target interval for slack reclamation as $[t_{cur},\ d_1(t_{cur}))$, which makes it possible for τ_k^O to finish its execution no later than $d_1(t_{cur})$. Then, considering the next earliest

Algorithm 1 EDF-Slack: scale decision for τ_k^O at t_{cur}

1: $p = 0$, U^M = the left-hand-side of Eq. (3)
2: **for** $i = n$ to 1, $\tau_i^M \in \{\tau_1^M, ..., \tau_n^M | d_1(t_{cur}) \leq \cdots \leq d_n(t_{cur})\}$ **do** {In reverse EDF order of mandatory sub-tasks}
3: $\quad U^M = U^M - \frac{C_i^M}{T_i}$
4: $\quad q_i = \max\left(0, RC_i^M - (1 - U^M) \cdot (d_i(t_{cur}) - d_1(t_{cur}))\right)$
5: $\quad U^M = \min\left(1.0, U^M + \frac{RC_i^M - q_i}{d_i(t_{cur}) - d_1(t_{cur})}\right)$
6: $\quad p = p + q_i$
7: **end for**
8: $Slack(t_{cur}, d_1(t_{cur})) = d_1(t_{cur}) - t_{cur} - p$
9: **return** $\arg\max_{s_k \in S_k^O}\{C_k^O(s_k) | C_k^O(s_k) \leq Slack(t_{cur}, d_1(t_{cur}))\}$

release of any mandatory sub-task is $d_1(t_{cur})$, it is impossible for a lower-priority optional sub-task to block the execution of any high-priority mandatory sub-task (although the execution of a higher-priority optional sub-task may delay the execution of some lower-priority mandatory sub-tasks). Therefore, we do not need to increase the blocking term (i.e., the first term in Eq. (3)), which allows to use Eq. (3) as a schedulability condition for EDF-Slack. This is achieved by exploiting the property of the S&M DNN execution model: while all sub-tasks are non-preemptive, there can be the execution of other sub-tasks between the execution of a mandatory sub-task and that of its corresponding optional sub-task.

On the other hand, achieving (i) entails a careful calculation of the remaining slack in $[t_{cur},\ d_1(t_{cur}))$. Since we do not know the size of RoI and actual execution time of a mandatory sub-task until it is completed, we need to assume that the resource demand for each future mandatory sub-task is up to C_k^M to guarantee no deadline miss. However, upon completion of a mandatory sub-task, we know its actual execution time corresponding to its actual RoI size and reclaim the unused portion of assigned resources that can be counted when calculating the slack time in $[t_{cur}, d_1(t_{cur}))$.

Alg. 1 presents details of the slack calculation and scale decision. At time t_{cur} when an optional sub-task τ_k^O is selected to be scheduled, we focus on the interval of $[t_{cur}, d_1(t_{cur}))$, where $d_i(t_{cur})$ is the i^{th} earliest absolute deadline after t_{cur} among all mandatory sub-tasks. In that interval, we examine all mandatory sub-tasks in reverse EDF order (the latest deadline first), where n is the number of tasks in τ (Line 2). Note that mandatory sub-tasks are indexed in EDF order (i.e., for τ_i^M and τ_k^M where $i < k$, $d_i(t_{cur}) \leq d_k(t_{cur})$). We consider that future mandatory sub-task invocations require the worst-case execution time and thus their utilization is the left-hand-side of Eq. (3) (line 1). We consider that the scheduler keeps track of the worst-case remaining execution time RC_i^M for the active mandatory sub-task of τ_i at t_{cur}; since τ_k^O is selected to be scheduled at t_{cur}, RC_i^M at t_{cur} is either 0 or C_i^M. We try to defer as much execution of mandatory sub-tasks as possible beyond $d_1(t_{cur})$ and compute the minimum amount of execution p that must execute before $d_1(t_{cur})$ in order to meet future deadlines of all mandatory sub-tasks (Lines 3–6). This step is repeated for all mandatory sub-tasks. For example, we calculate the maximum execution time of τ_n^M (whose deadline is $d_n(t_{cur})$) in $[d_1(t_{cur}),\ d_n(t_{cur}))$, which is $(1 - U^M) \cdot (d_n(t_{cur}) - d_1(t_{cur}))$ in Line 4; then, q_i in Line 4 is the minimum execution of τ_n^M in $[t_{cur}, d_1(t_{cur}))$. The execution rate of τ_n^M in $[d_1(t_{cur}),\ d_n(t_{cur}))$ is recorded in Line 5. This is similar to the approach in [36], [37]. Then, the slack is set to the remaining time slots except for p (i.e., the sum of q_i) over the interval $[t_{cur},\ d_1(t_{cur}))$ in Line 8. Therefore, when an optional sub-task τ_i^O is selected to be scheduled at time t_{cur}, its scale is determined according to Line 9.

As we designed, we can apply the schedulability analysis of EDF-MandFirst to EDF-Slack as follows.

Theorem 2: If Eq. (3) holds, τ is schedulable by EDF-Slack (i.e., it guarantees no sub-job deadline miss).

Proof: Due to the mechanism of determining the scale of optional sub-tasks for EDF-Slack, a lower-priority optional sub-task cannot block the execution of any high-priority mandatory sub-task. Therefore, we use the same blocking term as Eq. (3) (i.e., the first term).

Then, we apply the fact that there is no job deadline miss under EDF if the total utilization (including the blocking term) at any time is not greater than 1.0 [38]. Then, what we need to prove is EDF-Slack satisfies $U^M(t) + U^O(t) \leq 1.0$ for every t, where $U^M(t)$ and $U^O(t)$ denote the sum of run-time utilization of mandatory sub-tasks and optional sub-tasks, respectively, each of whose release time and deadline are before and after t, respectively. Under EDF-Slack, starting with setting the total static utilization of U^M in Line 1 of Alg. 1, the algorithm updates the run-time utilization by Lines 3–5, and calculates the largest q_i that does not compromise $U^M(t) + U^O(t) \leq 1.0$ in Line 4, and any optional sub-task performs its execution for at most $Slack(t_{cur}, d_1(t_{cur}))$ in $[t_{cur}, d_1(t_{cur}))$, implying that the theorem holds. ■

Run-time complexity. At each invocation (either job release or sub-task completion), our scheduling algorithm calculates the slack with the complexity of $O(1)$ for EDF-MandFirst and $O(n)$ for EDF-Slack in Alg. 1 where n is the number of tasks. Then, our algorithm determines the scale of an optional sub-task τ_i^O from S_i^O with the complexity of $O(1)$. Thus, the total complexity is $O(1)$ for EDF-MandFirst and $O(n)$ for EDF-Slack.

VI. Evaluation

A. Experimental Setup

Hardware and software. We have implemented and evaluated DNN-SAM on top of DarkNet [13], a representative ML framework, to enable seamless S&M execution in the existing object detection networks without any modification or retraining. The programmers can easily apply DNN-SAM to their DNNs in the same way they design DNNs without DNN-SAM. We conducted experiments on NVIDIA Jetson Xavier [24] running Ubuntu 18.04.4 and CUDA 10.0. Note that NVIDIA Jetson Xavier does not support CUDA multi-process service (MPS) and preemptive multitasking. We consider non-preemptive EDF where the executions are non-preemptive in a sub-task level. And, we utilized YOLOv3 [2]

TABLE I
EXECUTION TIME MEASUREMENT ON NVIDIA XAVIER

Time (ms)	$c_i^{\texttt{RoI}}$		$c_i^{\texttt{Split}}$		$c_i^{\texttt{Infer}}$		$c_i^{\texttt{Merge}}$
Average	2.4		4.0		38.4		0.1
Maximum	9.0		7.5		40.3		0.6
	$c_i^{\texttt{Infer}}(s_i)$						
Scale s_i	160	256	320	416	512	608	672
Average	31.3	38.4	69.3	97.8	115.6	182.7	213.4
Maximum	33.4	40.3	71.7	108.4	136.7	210.1	225.9

as a baseline network tested with 7,481 images on the KITTI in-vehicle camera dataset [25] to detect cars and pedestrians in the moderate level of difficulty. Note, YOLOv3 is a fully convolutional network that takes an input of arbitrary size and produces an output of the corresponding size. For RoI identification, we used the Velodyne LiDAR point clouds data (also available on the KITTI dataset) corresponding to the tested images. However, the KITTI dataset does not provide the corresponding IMU data, and hence we assume a constant vehicle speed. To evaluate the benefits of DNN-SAM in a realistic setup, we also implemented a real-time object detection system for emergency braking on a 1/10 scale self-driving car [39].

DNN execution time profiling and run-time overhead. DNN-SAM conducts DNN inferencing for mandatory and optional sub-tasks on a GPU, while the other modules including RoI identification, network split/merge and sub-task-scheduling use CPUs. We take a measurement-based approach to estimate the worst-case execution time of mandatory and optional sub-tasks, C_i^M and C_i^O. Our experiments measure the execution time of each component in C_i^M and C_i^O by running it 1,000 times, taking the maximum value among the measured as the WCET as summarized in Table I. For example, when a scale s_i of an input image varies from 160 to 672, the worst-case inference time ($c_i^{\texttt{Infer}}(s_i)$) is increased from 33.4ms to 225.9ms. The three key modules—RoI identification, network split, and network merge—in DNN-SAM take 2.4ms, 4.0ms, and 0.1ms on average, accounting for 1.1%, 1.8%, and 0.02% of the total execution time of a split-and-merge DNN execution pipeline with an optional sub-task's scale of 608, respectively. The total overhead of DNN-SAM constitutes only 2.9% of the total execution time, a relatively short delay compared to the inference time of the original DNN model. The additional memory required by DNN-SAM is about 2,361KB per task, constituting 0.1% of the total memory usage.

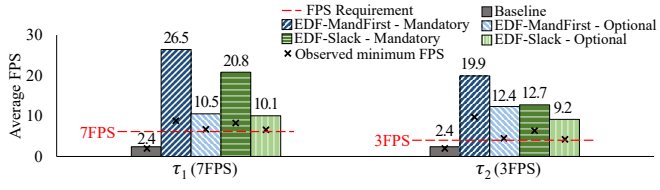
We also measure the overhead of our scheduling algorithms as a function of the number of DNN tasks (n). We run dummy DNN tasks for this measurement since at most four YOLOv3 networks can be loaded on the Xavier due to the memory constraint. When n is increased from 1 to 100, the run-time scheduling overhead increases from $26.4\mu s$ to $88.7\mu s$, which is almost linear in n and is not critical since it is five orders-of-magnitude shorter than the DNN execution time.

Approaches to be compared. We compare DNN-SAM with EDF-MandFirst and EDF-Slack against Baseline (vanilla DarkNet) in terms of inference latency and accuracy. We use two different metrics for inference accuracy: *accuracy of RoI* and *overall accuracy*. The accuracy of RoI is defined as the ratio of detected objects inside an RoI to the total ground truths, and the overall accuracy is that of all detected objects to the total ground truths. Frame-Per-Second (FPS) is used as the inference latency metric. Under EDF-MandFirst and EDF-Slack, DNN tasks go through their S&M DNN execution pipelines, and are scheduled under EDF-MandFirst and EDF-Slack, respectively. Baseline represents the state-of-the-art DNN inference framework, where DNN tasks process their corresponding unmodified networks, and are scheduled in a FIFO manner. Unless otherwise specified, Baseline uses the network size of 608×608, and DNN-SAM uses the maximum RoI size of $w = h = 256$ and network scales $S_i^O = \{0, 160, 256, 320, 416, 512, 608, 672\}$.

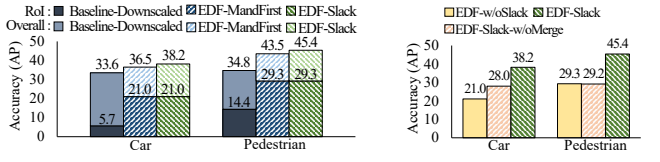
B. Evaluation Result

Effectiveness in terms of inference latency and accuracy. We would like to show the effectiveness of DNN-SAM in producing faster and more accurate response in RoI (safety-critical portion) while meeting all timing requirements for multiple DNN-based object detection tasks. In this experiment, we executed two object detection tasks with 3 and 7 FPS requirements, respectively, and measured their inference latency and accuracy metrics. Fig. 6a plots the FPS characteristics of mandatory sub-tasks and subsequent optional sub-tasks for EDF-Slack and EDF-MandFirst as well as the FPS characteristics of Baseline. The red dotted line represents the FPS requirements for the two tasks. Baseline shows 2.4 FPS on average for both tasks, i.e., both tasks cannot meet their FPS requirements. Such a low FPS on both tasks is observed because the ML framework handles DNN tasks with different timing constraints sequentially without prioritization. On the other hand, EDF-Slack and EDF-MandFirst show different FPS performance between mandatory and optional sub-tasks, i.e., higher FPS for mandatory sub-tasks than optional ones, while meeting all the FPS requirements for both tasks. For example, under EDF-MandFirst, mandatory sub-tasks produce 26.5 FPS and 19.9 FPS, while optional sub-tasks produce 10.5 FPS and 12.4 FPS for τ_1 and τ_2, respectively, satisfying both tasks' FPS requirements. Comparing to Baseline, EDF-MandFirst yields 9.7× and 4.8× faster inference results for RoI and an entire image, respectively, on average. Comparing to EDF-Slack, EDF-MandFirst achieves 1.4× faster inference results for RoI on average. This is because EDF-MandFirst is designed to statically prioritize mandatory sub-tasks over optional sub-tasks so as to detect objects in RoI as quickly as possible.

Fig. 6b compares the accuracy characteristics with the detailed breakdown of the accuracy of RoI (lower portion of the cumulative bar) and overall accuracy (total portion of the cumulative bar). Recall that, under DNN-SAM, the accuracy of RoI is calculated from the results of mandatory sub-tasks, and the overall accuracy is from the merged results of mandatory and optional sub-tasks. We observe that the inference results were obtained after their deadline under Baseline, which is useless for safety-critical applications. Therefore, to make a fair comparison with DNN-SAM, we compare Baseline-Downscaled that shares the same framework with Baseline but



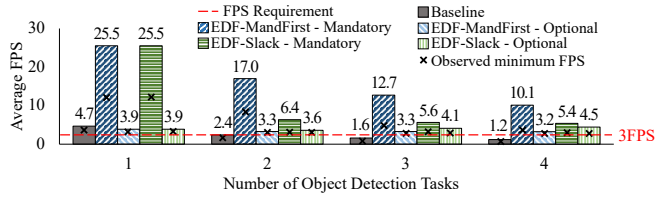
(a) FPS characteristics



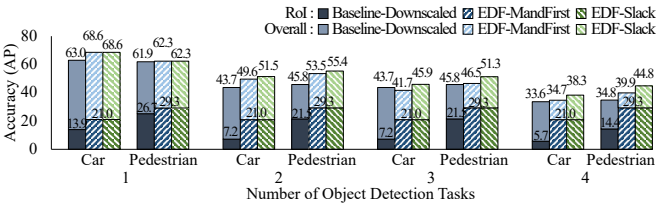
(b) Accuracy of RoI (lower portion) and overall accuracy (total portion)

(c) Overall accuracy: ablation study

Fig. 6. FPS and accuracy characteristics for object detection tasks.



(a) Average FPS characteristics



(b) Accuracy of RoI (lower portion) and overall accuracy (total portion)

Fig. 7. FPS and accuracy characteristics for increasing number of tasks.

processes DNN tasks with down-scaled networks so as to meet their FPS requirements under non-preemptive EDF. Baseline-Downscaled shows the overall accuracy of 33.6% and 34.8% and the accuracy in RoI of 5.7% and 14.4% for detecting cars and pedestrians, respectively. On the other hand, DNN-SAM with EDF-Slack shows much higher accuracy in RoI of 21.0% and 29.3% (3.7× and 2.0× RoI accuracy improvement) than Baseline-Downscaled for detecting cars and pedestrians, respectively, while showing comparable (or improved) overall accuracy, i.e., 38.2% and 45.4%. Such high accuracy in RoI can be interpreted as the benefit of S&M DNN execution, i.e., processing the safety-critical region with the original resolution. In addition, EDF-Slack shows higher overall accuracy than EDF-MandFirst. EDF-Slack shows the overall accuracy of 38.2% and 45.4% for car and pedestrian detection, respectively, while EDF-MandFirst shows the overall accuracy of 36.5% and 43.5%. This is because EDF-Slack is designed to secure more resources for optional sub-tasks by assigning priorities to sub-tasks by the EDF policy without separating mandatory and optional sub-tasks.

One may wonder how much portion each component in DNN-SAM contributes to maintaining comparable overall accuracy while achieving high accuracy in RoI. To demonstrate this, we also compare EDF-w/oSlack and EDF-Slack-w/oMerge with EDF-Slack as shown in Fig. 6c. EDF-w/oSlack schedules sub-tasks in the same way as EDF-Slack but the scales of optional sub-tasks are determined statically and offline according to the schedulability analysis in Eq. (3). Without run-time slack reclamation, EDF-w/oSlack cannot execute any optional sub-task with a scale larger than 0 since the left-hand-side of Eq. (3) is close to 1.0, yielding a large overall accuracy drop by 17.2 percentage points (%p) and 16.1%p relative to EDF-Slack for detecting cars and pedestrians, respectively. EDF-Slack-w/oMerge uses the same scheduling and scale decision algorithm with EDF-Slack but skips the merge operation, i.e., the overall accuracy is determined solely by the inference results of optional sub-tasks. Without the network merge module in DNN-SAM, EDF-Slack-w/oMerge shows an overall accuracy drop by 10.2%p and 16.2%p relative to EDF-Slack for detecting cars and pedestrians, respectively. Whereas, EDF-Slack effectively merges the inference results of mandatory and optional sub-tasks into a complete one and adaptively selects the scales of optional sub-tasks by utilizing available slack at run-time, achieving higher overall accuracy. DNN-SAM with EDF-MandFirst shows similar accuracy results as the case of EDF-Slack.

In summary, DNN-SAM not only produces faster and more accurate inference results for safety-critical portions of input images but also achieves the overall accuracy comparable to the existing ML framework using the original DNN models, while meeting all timing requirements.

Evaluation with an increasing number of tasks. Next, we evaluated how our framework effectively handles an increasing number of tasks in a task set. In this experiment, we conducted a set of experiments while varying the number of tasks (n) from one to four, each of which has 3 FPS requirement. Fig. 7a shows the average FPS for different numbers of object detection tasks running on the system. Under Baseline, a task set with only one task can meet its FPS requirement while other task sets with more than one task cannot (the gray bar is below the FPS requirement). On the other hand, DNN-SAM with EDF-Slack and EDF-MandFirst satisfies the FPS requirement for all the cases of $n = 1, 2, 3$ and 4. As the number of tasks increases, the available slack resource for optional sub-tasks decreases. EDF-Slack and EDF-MandFirst adaptively assign smaller scales to optional sub-tasks as the number of tasks increases to meet the FPS requirement. In addition, EDF-MandFirst shows higher average FPS results for mandatory sub-tasks than EDF-Slack, e.g., 10.1 and 5.4 with EDF-MandFirst and EDF-Slack, respectively, when $n = 4$. This is because EDF-MandFirst is designed to secure mandatory sub-tasks from the potential interference of optional sub-tasks by assigning statically higher priorities to mandatory sub-tasks than optional ones.

Fig. 7b shows RoI accuracy and overall accuracy. Note that Baseline cannot meet the FPS requirement for $n \geq 2$, so we compare Baseline-Downscaled as in Fig. 6b (Note that the accuracy result of Baseline-Downscaled is equivalent to that of Baseline when $n = 1$). As the number of tasks increases

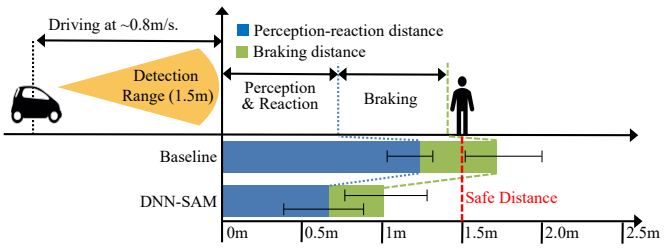


Fig. 8. Evaluation with Emergency Braking System

from 1 to 4, EDF-Slack and EDF-MandFirst exhibit consistently high RoI accuracy of 21.0% and 29.3% for car and pedestrian detection, respectively, while Baseline-Downscaled's RoI accuracy is decreased from 13.9% and 26.7% to 5.7% and 14.4%, respectively. Such a result of sustaining high RoI accuracy regardless of n can be interpreted as the benefit of S&M DNN execution that enables to process safety-critical portions with the original resolution. Moreover, EDF-Slack exhibits higher overall accuracy than Baseline-Downscaled for all values of the number of tasks. For example, when $n = 4$, EDF-Slack shows the overall accuracy of 38.3% and 44.8% for car and pedestrian detection, respectively, while Baseline-Downscaled shows the overall accuracy of 33.6% and 34.8%. Such improvements mainly come from i) adaptive scaling of optional sub-tasks by efficient use of slack resources at run-time while meeting the FPS requirements and ii) effective merging the outputs of mandatory and optional sub-tasks.

In summary, DNN-SAM with EDF-Slack and EDF-MandFirst can be adapted to an increasing number of tasks using adaptive scaling while EDF-Slack and EDF-MandFirst effectively achieve their own objectives.

C. Case Study: Emergency Braking

Finally, DNN-SAM has been deployed into a 1/10 scale self-driving car equipped with an NVIDIA Jetson Xavier to perform a case study of emergency braking.[5] We executed two object detection tasks for front/back cameras both at 5 FPS, and the braking is activated as soon as an object is detected in RoI of an image from the front camera. In this experiment, a car is moving toward the pedestrian at $0.8m/s$, and the camera's field-of-view is limited to $1.5m$, i.e., the required braking distance is limited to that range. Fig. 8 shows the average stopping distances (plotted as bar) with a breakdown of the distance up to the detection point (perception-reaction) and the actual braking distance (braking) as well as the maximum/minimum distances (error bar) out of 15 trials with and without using DNN-SAM. With Baseline, a stopping decision is made at the $1.2m$ spot and stops at the $1.7m$ spot on average, thus exceeding the safety distance. The worst-case stopping distance is $2.1m$ which is far beyond the safety distance. In contrast, DNN-SAM makes a stopping decision at the $0.7m$ spot, and stops at the $1.0m$ spot on average within the safety distance. The distance up to the detection is reduced by 1.9× compared to Baseline. The worst-case stopping distance is $1.3m$ still within the safety distance, demonstrating enhanced safety and quality of control in a real-world environment.

[5]See https://rtcl.dgist.ac.kr/index.php/dnnsam/ for the demo video of emergency braking.

VII. Conclusion

We have focused on distinct requirements of real-time DNN-based object detection in autonomous cars, that is providing different levels of detection quality to image portions with different criticality levels while guaranteeing all timing constraints. To this end, we have developed DNN-SAM, a dynamic split-and-merge DNN execution and scheduling framework, which supports i) split and merge interfaces to transparently decompose an original DNN into two sub-tasks and ii) a lightweight real-time scheduler to prioritize mandatory sub-tasks over optional ones with adaptive selection of the scales of optional sub-tasks. In future, we aim to identify a safety-critical portion of each image more efficiently but accurately by considering diverse driving contexts (e.g., vehicle's trajectory) to further enhance the utility and power of DNN-SAM.

Acknowledgement

This work was supported in part by (1) the National Research Foundation of Korea (NRF) grant (2017M3A9G8084463, 2018R1A5A1060031 (ERC), 2020R1F1A1076058, 2021R1A4A1032252, 2021R1A2B5B02001758, 2021K2A9A1A01101570), (2) Institute of Information & communications Technology Planning & Evaluation (IITP) grant (2020-0-101741: Grand ICT Research Center support program) funded by the Korea government (MSIT), (3) the DGIST R&D Program of MSIT (20-CoE-IT-01), and (4) US National Science Foundation under Grant CNS-1646130.

References

[1] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *ECCV*, 2016.
[2] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NeurIPS*, 2015.
[4] R. Girshick, "Fast R-CNN," in *ICCV*, 2015.
[5] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan, and R. Fan, "Fully quantized network for object detection," in *CVPR*, 2019.
[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, 2017.
[7] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed optimisation for non-convex problems," in *ICML*, 2018.
[8] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.
[9] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *NeurIPS*, 2016.
[10] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *CVPR*, 2017.
[11] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.
[12] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2017.
[13] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[14] E. Kim, C. Ahn, and S. Oh, "NestedNet: Learning nested sparse structures in deep neural networks," in *CVPR*, 2018.
[15] J.-E. Kim, R. Bradford, and Z. Shao, "AnytimeNet: controlling time-quality tradeoffs in deep neural network architectures," in *DATE*, 2020.
[16] S. Lee and S. Nirjon, "SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *RTAS*, 2020.
[17] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *RTAS*, 2020.
[18] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *RTSS*, 2018.
[19] H. Zhou, S. Bateni, and C. Liu, "S^{3}DNN: Supervised streaming and scheduling for gpu-accelerated real-time dnn workloads," in *RTAS*, 2018.
[20] 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 *RTCSA*, 2020.
[21] 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 *RTAS*, 2019.
[22] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *RTSS*, 2019.
[23] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *RTSS*, 2020.
[24] Jetson AGX Xavier Developer Kit. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
[25] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI Vision Benchmark Suite," in *CVPR*, 2012.
[26] J. Kocić, N. Jovičić, and V. Drndarević, "Sensors and sensor fusion in autonomous vehicles," in *Telecommunications Forum*, 2018.
[27] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *IROS*, 2016.
[28] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.
[29] A. Martinelli, "Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination," *IEEE Transactions on Robotics*, 2012.
[30] 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.
[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.
[32] J. Wang, L. Song, Z. Li, H. Sun, J. Sun, and N. Zheng, "End-to-end object detection with fully convolutional network," *arXiv preprint arXiv:2012.03544*, 2020.
[33] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[34] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 111–130, 2001.
[35] T. Baker, "A stack-based resource allocation policy for realtime processes," in *RTSS*, 1990.
[36] P. Pillai and K. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *SOSP*, 2001.
[37] H. S. Chwa, K. G. Shin, H. Baek, and J. Lee, "Physical-state-aware dynamic slack management for mixed-criticality systems," in *RTAS*, 2018.
[38] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes," in *RTSS*, 2003.
[39] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.