Received 22 April 2024, accepted 8 May 2024, date of publication 17 May 2024, date of current version 29 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3402239

RESEARCH ARTICLE

Optimizing Monocular Driving Assistance for Real-Time Processing on Jetson AGX Xavier

HUY-HUNG NGUYEN, DUONG NGUYEN-NGOC TRAN, (Member, IEEE), LONG HOANG PHAM, (Member, IEEE), AND JAE WOOK JEON, (Senior Member, IEEE)
Department of Electrical and Computer Engineering, College of Information and Communication Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Jae Wook Jeon (jwjeon@skku.edu)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by Korean Government (MSIT), an intelligent system for 24/7 real-time traffic surveillance on edge devices, under Grant 2021-0-01364.

ABSTRACT While computer vision and computing technology advances have facilitated advanced driver assistance applications, systems with multi-task design remain highly demanding to operate at high speed on resource-constrained devices. Our study addresses this challenge by proposing a real-time driver assistance solution specifically developed for a single Jetson AGX Xavier embedded device. It simultaneously performs lane detection, traffic object detection and recognition, and rule-based scene analysis. To achieve high throughput (up to 43 frames per second) without reliance on additional hardware or cloud server, the system exploits Jetson device's specialized AI accelerator and employs various optimization techniques: multithreaded programming, the TensorRT framework, and post-training quantization. The modular design integrates state-of-the-art task-specific methods and ensures adaptation to diverse traffic scenarios across countries as well as future hardware and solutions. Experimental results using a Korean dashcam traffic dataset validated the system's effectiveness and practicality.

INDEX TERMS Advanced driver assistance system (ADAS), autonomous driving, edge device, real-time.

I. INTRODUCTION

An advanced driver assistance system (ADAS) deploys a combination of artificial intelligence technologies in a modern vehicle to enhance driver safety and awareness. Through a human–machine interface, an ADAS can give vehicle drivers essential information about current traffic conditions such as road and lane boundaries and traffic lights and signs and provide warnings about possible hazards for collision avoidance.

In recent years, different aspects of ADAS for autonomous vehicles have attracted research interest for their potential role in reducing the number of accidents, injuries, and fatalities on the road [1]. Substantial progress has been made, especially in using deep learning for computer vision tasks such as object detection and lane recognition. Although state-of-the-art (SOTA) algorithms often rely on complex and computationally demanding network models, many research studies have focused on creating compact and efficient models capable of delivering real-time performance. In addition to software development, progressively more powerful embedded platforms targeting autonomous driving have been launched or planned, including NVIDIA Jetson and Drive, Intel Mobieye EyeQ, Tesla Hardware [2], [3]. Moreover, specialized hardware called AI accelerators or neural processing units (NPUs) have been created to accelerate deep learning operations with high power efficiency. The Google Tensor Processing Unit and NVIDIA Deep Learning Accelerator (DLA) [2], [4] are prime examples of this approach.

However, building autonomous vehicles with the most powerful embedded platform and every sensor would be incredibly expensive and potentially underutilized. Conversely, integrating every possible feature into a system may lead to excessive computational costs, rendering simultaneous real-time operation unfeasible. Therefore, to achieve widespread adoption, it is important and still challenging to develop an ADAS with an appropriate collection of features

The associate editor coordinating the review of this manuscript and approving it for publication was Byoung Wook Choi.

that can perform efficiently and effectively on affordable, resource-constrained edge devices.

This study proposes a real-time driving assistance system on edge device that uses traffic images from a monocular dashboard camera mounted on the vehicle. The system has a modular architecture with modules implementing SOTA solutions for lane detection, traffic object detection and recognition (vehicles, pedestrians, traffic lights, traffic signs, road markings), and ruled-based scene analysis for collision avoidance. Inspired by the feature set in [5], the modules provide the core functionality of a standard ADAS, as outlined in [1].

While deploying each task individually in real-time on an edge device is generally achievable [6], [7], [8], [9], multi-task frameworks are still highly demanding [10], [11], [38]. Existing approaches to overcome this typically involve either distributed computing with multiple edge devices [5], or cloud-based offloading with an onboard edge device and a remote cloud server [12], [13], [14]. Both approaches necessitate additional management of cross-device or edge-cloud communication and data transfer over a stable, low-latency and secure connection. This requirement introduces considerable complexity in development and maintenance.

Our proposed approach focuses on implementing the system exclusively on a single Jetson AGX Xavier embedded board. All the available computing resources on the device, the multi-core central processing unit (CPU), the graphics processing unit (GPU), and especially the two DLA cores, are leveraged to handle simultaneous execution of both deep learning and non–deep learning tasks. Furthermore, we analyze the workload of the deep learning tasks and strategically allocate computing units to each task. Several optimization techniques are then applied to enable real-time performance on the edge device: multithreaded processing, the NVIDIA TensorRT inference framework and post-training quantization. Our modular architecture not only enhances scalability to more robust hardware and upgradability to future task-specific solutions, but also ensures adaptability across diverse traffic scenarios and regulations in various countries. Most importantly, this single-device approach reduces installation and maintenance costs as well as overall power consumption, and eliminates the need to address potential cross-device communication issues affecting latency and stability. We evaluated the system through extensive experiments on a Korean dashcam traffic dataset from KATECH (Korea Automotive Technology Institute) [15], [16] using the targeted Jetson AGX Xavier, the latest Jetson AGX Orin, and a desktop system. The empirical results demonstrate the system's effectiveness and real-world viability.

In brief, our key contributions are:

- Establish a multi-task ADAS pipeline using a monocular camera. The system simultaneously performs lane detection, traffic object detection and recognition, and rule-based scene analysis for collision avoidance.
- Design a modular framework, where each module handles a specific main task. The modular design minimizes the effort required for adaptation to alternative hardware, future task-specific solutions and diverse traffic conditions in various countries.
- Construct the framework to work solely on a single Jetson AGX Xavier board. All available computing components, especially the two DLA cores, are utilized to operate deep learning and non-deep learning tasks.
- Optimize the framework to perform in real-time at up to 43 frames per second (FPS) on the edge device. To achieve that performance, tasks are assigned computing units sufficient for their workload. Multithreaded programming, the TensorRT inference and post-training quantization techniques are also applied.
- Analyze the experiment results from different system configurations on the target and other platforms to prove our design reasons and to show the extensibility of the proposed framework.

The rest of this paper is organized as follows. Section II discusses related work on ADAS. Section III describes the proposed system's features and implementation. The experiments are examined in Section IV, and the conclusions are presented in Section V.

II. RELATED WORK

This section briefly reviews common deep learning–based solutions for lane detection and traffic object detection tasks outside ADAS. After that, we describe the main approaches to ADAS.

A. TASK SPECIFIC IMPLEMENTATION

1) SINGLE-TASK APPROACH

a: OBJECT DETECTION

Object detection methods can be broadly categorized into two main approaches. Two-stage detectors, exemplified by the R-CNN method and its variants [17], [18], [19], [20], typically employ selective search or a region proposal network to generate candidate object locations and then use a classifier to refine and categorize them. These methods usually achieve high accuracy, but they require great processing power.

One-stage detectors, led by the YOLO series [21], [22], skip the region proposal and directly predict objects from the input image. They can use either predefined anchor boxes placed across the image grid [21], [22], [23], or a feature map [24], [25], [26] to regress the bounding boxes. These anchor-based and anchor-free detectors have significantly faster processing speed, but they are somewhat less accurate than their two-stage counterparts. Recent research has tried to bridge that gap by using various techniques to improve both speed and precision [23], [26].

In this study, we adopt one-stage YOLOv8 [26] as a detector for multiple traffic objects, due to its SOTA performance in both accuracy and processing time.

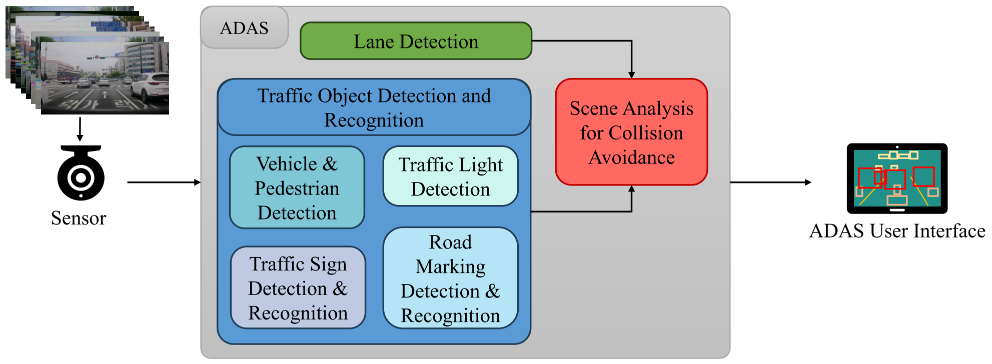
ADAS
Lane Detection
Traffic Object Detection and Recognition
Vehicle & Pedestrian Detection
Traffic Light Detection
Traffic Sign Detection & Recognition
Road Marking Detection & Recognition
Scene Analysis for Collision Avoidance
Sensor
ADAS User Interface

FIGURE 1. General feature design. There are three main modules: Lane detection, traffic object detection and recognition, which includes four sub-modules corresponding to four traffic object categories, and scene analysis for collision avoidance.

b: LANE DETECTION

There are generally three main approaches to lane detection. The first approach is based on pixel-wise instance segmentation, which maps an input image to an output mask, assigning to each pixel a label indicating whether or not it belongs to a lane line [27], [28]. These methods are pixel-wise precise, and they can detect infinite lanes without explicitly defining them. However, they have very low processing speed.

Second is the anchor-based approach, which is derived from the anchor-based object detection pipeline that regresses the deviation (e.g., position, shape, size) of the predefined line or curve anchors that cover potential locations in the image, aligning them with the actual lanes [29], [30], [31]. The main advantages of these methods are their high speed, making them suitable for real-time applications, and their ability to handle complex lane shapes and multiple lanes. The drawback is that their performance depends on the optimization of the anchor design and placement on the image.

The third approach represents the lane boundaries as a parametric model, for example, a polynomial or a spline, and then regresses its coefficients [32]. Methods following this approach are robust to noise and occlusions, and they can fit smooth and continuous lane curves at high processing speed. However, the choice of the polynomial's degree after the assumption of a specific form of lane model affects the performance and might not capture the real shape of the lanes.

In this study, we choose to implement anchor-based Ultra Fast Lane Detection v2 (UFLDv2) [31] as the SOTA method for lane detection.

2) MULTI-TASK APPROACH

The multi-task or joint-task learning approach involves simultaneously solving for multiple tasks, such as traffic object detection, drivable area segmentation, and lane detection, using a single neural network. The feature encoder is shared among all tasks, and a set of task-specific heads produces the final output. In addition, the overall loss function is a weighted sum of multiple losses corresponding to the tasks. Some examples of multi-task learning for autonomous driving are MultiNet [33], DLT-net [34], YOLOP [35], YOLOPv2 [36], and HybridNets [37].

This approach has several advantages over training separate networks for each task. It reduces the computational overhead by avoiding redundant feature extraction. It also allows the tasks to learn from one another, which potentially improves their accuracy. Moreover, it does not require the execution of multiple networks, which simplifies direct deployment.

However, this approach also has some challenges. First, it increases the complexity of the network design. Next, it can suffer from negative transfer, that is, the gradient updates for the different tasks can conflict with one another, or some tasks can dominate others during training. Thus, to improve the performance of one task, it can be easier, more effective, and less time-consuming to train the related model separately, so any tricks applied do not affect the other tasks, rather than training the whole multi-task model together. Third, resource-optimized deployment on edge devices can get more complicated than the distributed execution of separate networks on different computing components, particularly the GPU and NPUs.

In our proposed method, we decide not to use the multi-task approach due to the difficulty of adapting certain branches of the network to different countries' traffic conditions as well as complications in optimization for edge devices. Instead, we establish a modular framework integrating task-specific modules with SOTA solutions, which has the advantages of scalability, upgradability and adaptability.

B. ADAS IMPLEMENTATION

Different approaches have been proposed to implement a multi-task ADAS. The first approach performs all ADAS functions on a single computing system [38]. This has some benefits, such as minimizing data transfer delays for better responsiveness and simplifying the development and testing process. The disadvantage is that high-speed performance will demand high processing power, high power consumption, and high installation costs. As reported in [10], it requires a high power desktop-grade GPU to operate an entire ADAS with 2 to 3 tasks. Hence, these systems are quite difficult to deploy broadly, especially on budget vehicles.

Another approach uses a distributed computing system, with each ADAS task performed by one of several dedicated edge devices [5]. This approach does not require a particularly high-performance system, potentially lowering the vehicle cost. Also, the ADAS can be designed to be modular and scalable, so it is much simpler to upgrade an existing feature or add a new functionality to the system. However, this requires that stable internal cross-device communication be maintained to manage data transfer latency, quality, and security, which adds complexity when developing the system. As a result, real-time performance is not always guaranteed, as demonstrated by [5]. The maintenance of multiple devices is also relatively complicated and time-consuming.

The third approach for implementing ADAS uses a cloud-based offloading system [12], [13], [14]. It is a hybrid design by which low computational tasks are performed by the onboard edge device, and computationally intensive tasks are offloaded to a remote cloud server, which usually has powerful resources, for processing. Such a design does not require that a powerful embedded device be installed in the vehicle, and cloud-based features can be updated and improved without requiring in-vehicle hardware changes. On the other hand, the system relies on a stable and reliable internet connection, which might not be available in all situations, and introduces high data latency. The experiments in [13] and [14] show that the edge-cloud data transfer step consume the majority of per-frame processing time. Optimizing such communications, especially at a large-scale, can be extremely complicated in real-world applications.

In our proposed method, by combining the first and second approaches, we implement and optimize our system to fully operate on a single Jetson AGX Xavier embedded platform by utilizing all provided computing units: the CPU, GPU and two DLA cores. This approach retains the advantages of both approaches while eliminating the need for handling complicated cross-device communication and data transfer.

III. PROPOSED METHOD

A. FEATURE DESIGN

Our proposed system can be divided into three main modules: Lane Detection, Traffic Object Detection and Recognition, and Scene Analysis for Collision Avoidance. Of these modules, Traffic Object Detection and Recognition is the most important. It consists of four sub-modules: Vehicle and Pedestrian Detection, Traffic Light Detection, Traffic Sign Detection and Recognition, and Road Marking Detection and Recognition. These modules and sub-modules provide information about the traffic scene that is essential for driving assistance. Fig. 1 shows the general feature design of the system.

1) LANE DETECTION

Road lane detection enables the car to align itself correctly on the road and helps the driver keep in the ego-lane and monitor

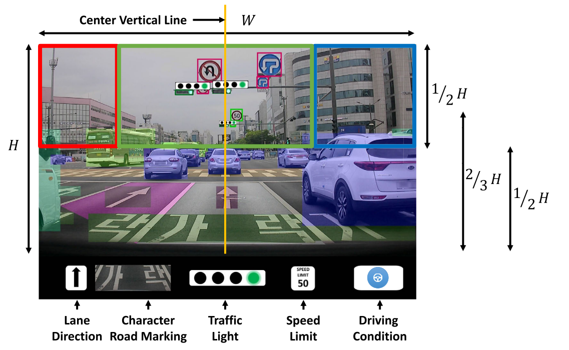

FIGURE 2. System user interface. The top traffic frame has detection results visualized. The bottom panel displays important traffic condition information. The red, green and blue boxes are the priority regions used in traffic light and sign detection.

TABLE 1. Traffic object classes.

Categories	Class
Vehicle & pedestrian	car
	bus
	truck
	motorcycle
	bicycle
	pedestrian
Traffic light	red
	yellow
	green
	red & arrow
	green & arrow

Categories	Class
Traffic sign	speed limit
	other
Road marking	straight arrow
	left arrow
	right arrow
	u-turn arrow
	straight left arrow
	straight right arrow
	other arrow
	crosswalk
	number
	character

lane departures. It also facilitates the recognition of relevant road markings in subsequent stages of the system. To speed up the processing of the UFLDv2 method, we cropped the image and input only the lower half of the traffic frame to the network, assuming that the upper half consists mainly of objects irrelevant to lane detection, such as traffic signs, traffic lights, building, and sky.

The result of lane detection is a set of lane lines:

$$L = \{l_1, l_2, \ldots\} \tag{1}$$

where each line l_i is presented by a set of key points:

$$p = (x, y) \tag{2}$$
$$l_i = \{p_{i,1}, p_{i,2}, \ldots\}. \tag{3}$$

The ego-lane can be identified by finding the two lane lines that are closest to the center vertical line of the camera frame, one on each side. The center vertical line is defined as:

$$l_C = \{p_{C,k} | x_{C,k} = {}^W/_2\} \tag{4}$$

where W is the width of the traffic frame. In Fig. 2 this line is orange.

For each l_i, we have:

$$l_{i,\text{left}} = \{p_{i,j} | p_{i,j} \in l_i, x_{i,j} < {}^W/_2\} \tag{5}$$
$$l_{i,\text{right}} = \{p_{i,j} | p_{i,j} \in l_i, x_{i,j} > {}^W/_2\} \tag{6}$$

where $l_{i,\text{left}}$ and $l_{i,\text{right}}$ are subsets of l_i containing points on the left and right of the center vertical line, respectively. Thus,

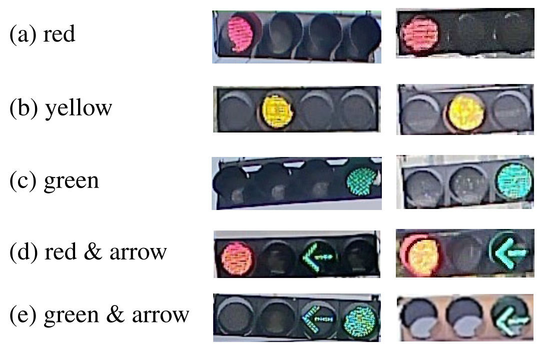

FIGURE 3. Examples of traffic lights' variants.

if $|l_{i,\text{left}}| > |l_{i,\text{right}}|$, the lane line l_i is on the left of the center vertical line, and vice versa.

The Hausdorff distance is used to estimate the distance between each lane line and the center vertical line:

$$\begin{aligned} d_H(l_i, l_C) = \max\{ &\max_{p_{i,j}\in l_i} \min_{p_{C,k}\in l_C} \|p_{i,j} - p_{C,k}\|, \\ &\max_{p_{C,k}\in l_C} \min_{p_{i,j}\in l_i} \|p_{C,k} - p_{i,j}\|\}. \end{aligned} \tag{7}$$

The ego-lane is then defined as:

$$\begin{aligned} L_{\text{ego}} &= (l_{\text{ego-left}}, l_{\text{ego-right}}) \\ &= (\operatorname*{argmin}_{l_i\in L_{\text{left}}} d_H(l_i, l_C), \operatorname*{argmin}_{l_j\in L_{\text{right}}} d_H(l_j, l_C)) \end{aligned} \tag{8}$$

where L_{left} and L_{right} are subsets of L containing lines on the left and right of the center vertical line, respectively.

2) TRAFFIC OBJECT DETECTION AND RECOGNITION

This module is responsible for detecting objects from different categories in the traffic scene with YOLOv8. After the detection bounding box results are obtained, we split them into four categories: (1) vehicles and pedestrians, (2) traffic signs, (3) traffic lights, and (4) road markings. Each category is then fed into the corresponding post-processing sub-module to identify the most relevant information. Table 1 lists all of the traffic object classes.

a: VEHICLE AND PEDESTRIAN DETECTION

It is vital for a practical driving assistance system to be able to accurately detect the vehicles and pedestrians in the camera field of view to prevent possible accidents. There are six classes to be detected: car, bus, truck, motorcycle, pedestrian, and bicycle. To further optimize processing efficiency and focus on the most relevant information, we keep only the bounding boxes from the lower two-thirds of the captured frame. This targeted approach effectively filters out extraneous detections of objects outside the designated road region because they are unlikely to be part of the immediate traffic situation.

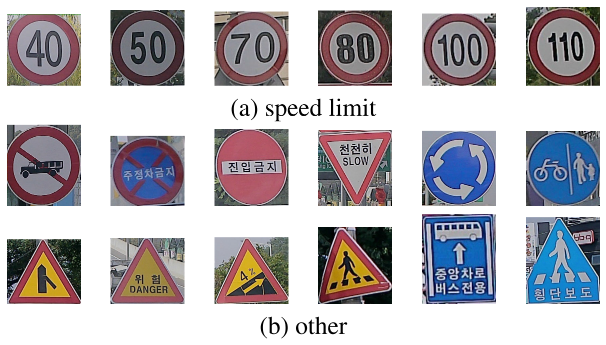

FIGURE 4. Examples of traffic signs.

b: TRAFFIC LIGHT DETECTION

Fig. 3 shows the different states of two variants of traffic lights in the KATECH dataset, with and without an arrow signal. At certain parts of the road, especially at intersections, it is possible to have several traffic lights in the camera's field of view that are responsible for different lanes and directions. Four constraints are applied to identify the correct traffic light.

- First, only a traffic light detected within the top half of the camera's view is processed, with the assumption that the system should detect the light as soon as possible while it is still relevant, and the bottom half of the traffic frame contains mostly vehicles, pedestrians, and the road region. This choice also speeds the processing time.
- Second, the upper half of the traffic frame is split horizontally into three regions with a predefined ratio, determined from the analysis of traffic sign position in the complete training dataset, following [40]. In Fig. 2, the red, green, and blue boxes represent the corresponding left, middle, and right regions, respectively. There are three priority levels, with the middle region having the highest priority, followed by the right region and then the left region. The priority levels are set to identify which traffic light is relevant to the current vehicle. To determine which region a traffic light bounding box belongs to, we determine whether its center point is within the region:

$$\text{bbox}_{\text{light},i} = (x_{i,\min}, y_{i,\min}, x_{i,\max}, y_{i,\max}) \tag{9}$$

$$\begin{aligned} \text{center}_i &= (x_{i,c}, y_{i,c}) \\ &= (\frac{x_{i,\min} + x_{i,\max}}{2}, \frac{y_{i,\min} + y_{i,\max}}{2}) \end{aligned} \tag{10}$$

$$\text{region}_j \in \{\text{region}_{\text{left}}, \text{region}_{\text{middle}}, \text{region}_{\text{right}}\} \tag{11}$$

$$\text{center}_i \in \text{region}_j \tag{12}$$

- Third, inside each region, the priority of traffic lights decreases vertically from top to bottom because, as a traffic light gets closer to the vehicle, it appears higher in the captured traffic image; and the closer the light is to the vehicle, the more relevant it is.
- Fourth, to reduce ambiguous cases such as occlusion causing incorrect identification, only traffic lights that

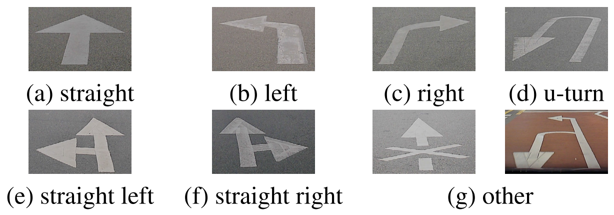

FIGURE 5. Examples of arrow road markings.

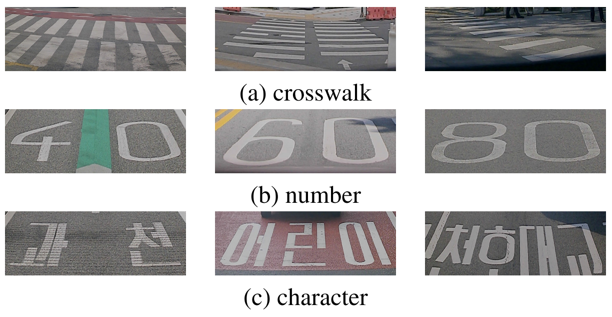

FIGURE 6. Examples of non-arrow road markings.

are fully visible are considered, especially those that are detected at the edges of the traffic frame.

Therefore, the dominant traffic light can be defined as:

$$\text{priority}_{\text{light},i} = (\text{priority}_{\text{region}_j,\text{center}_i\in\text{region}_j}, \text{priority}_{y_{i,c}}) \tag{13}$$

$$\text{bbox}_{\text{light,main}} = \text{argmax priority}_{\text{light},i}. \tag{14}$$

c: TRAFFIC SIGN DETECTION AND RECOGNITION

Different countries have unique designs for traffic signs. In the KATECH dataset, we classify Korean traffic signs into two main classes: speed limit and other. The speed limit signs are usually circular with a red border and a large number in the center, indicating the maximum speed allowed on the next section of the road. The other signs, with varied shapes and colors, are used to give warnings, prohibitions, and mandatory instructions. Fig. 4 depicts the traffic signs contained in the dataset.

The process starts by applying the same four constraints as in the previous subsection (III-A2b) to identify the traffic sign with the highest priority:

$$\text{bbox}_{\text{speedsign},i} = (x_{i,\min}, y_{i,\min}, x_{i,\max}, y_{i,\max}) \tag{15}$$

$$\begin{aligned}\text{center}_i &= (x_{i,c}, y_{i,c}) \\ &= (\frac{x_{i,\min}+x_{i,\max}}{2}, \frac{y_{i,\min}+y_{i,\max}}{2})\end{aligned} \tag{16}$$

$$\text{region}_j \in \{\text{region}_{\text{left}}, \text{region}_{\text{middle}}, \text{region}_{\text{right}}\} \tag{17}$$

$$\text{priority}_{\text{speedsign},i} = (\text{priority}_{\text{region}_j,\text{center}_i\in\text{region}_j}, \text{priority}_{y_{i,c}}) \tag{18}$$

$$\text{bbox}_{\text{speedsign,main}} = \text{argmax priority}_{\text{speedsign},i}. \tag{19}$$

For other signs, the detection bounding boxes are sufficient. For the speed limit signs, we apply the LPRNet model [41], pretrained on Korean data, on their cropped region of interest (ROI) to recognize the upcoming maximum speed limit that the vehicle needs to follow. This network was chosen for its high performance in both accuracy and processing speed. It was also verified by NVIDIA to work well on different Jetson models [42].

d: ROAD MARKING DETECTION AND RECOGNITION

Road markings are another important part of the traffic scene because they provide guidance and direction to the vehicle driver. The proposed system focuses on main arrow road markings, which indicate the direction of the lane they are painted on: straight, left, right, straight left, straight right, u-turn, and other. Crosswalk, number, and character road markings are also detected and localized by the system. Similar to speed limit traffic signs, number road markings show the maximum speed limit allowed in a particular lane. Likewise, character lane markings provide guidance such as upcoming school zone or children/elderly's protection zone, so the vehicle driver can adjust their driving appropriately. The markings are shown in Fig. 5 and Fig. 6.

To identify the road markings relevant to the vehicle, we keep only those that belong to the ego-lane, that is, those whose center point stays within the region created by the left and right ego-lane lines:

$$\text{bbox}_{\text{roadmarking},i} = (x_{i,\min}, y_{i,\min}, x_{i,\max}, y_{i,\max}) \tag{20}$$

$$\begin{aligned}\text{center}_i &= (x_{i,c}, y_{i,c}) \\ &= (\frac{x_{i,\min}+x_{i,\max}}{2}, \frac{y_{i,\min}+y_{i,\max}}{2})\end{aligned} \tag{21}$$

$$\text{center}_i \in \text{region}_{\text{ego-lane}}. \tag{22}$$

This constraint will also help reduce the computational cost of the post-processing step.

Next, the priority of road markings decreases vertically from bottom to top. It is because as a road marking gets closer to the vehicle, it appears lower in the captured traffic image; and the closer the road marking is to the vehicle, the more relevant it is:

$$\begin{aligned}\text{bbox}_{\text{roadmarking,main}} &= \text{argmax priority}_{\text{roadmarking},i} \\ &= \text{argmax priority}_{y_{i,c}} = \text{argmax}\, y_{i,c}.\end{aligned} \tag{23}$$

To recognize the speed limit in the detected number road markings, we feed their cropped ROI to the same LPRNet model mentioned in the previous subsection (III-A2c).

3) SCENE ANALYSIS FOR COLLISION AVOIDANCE

As the system's final stage, this module synthesizes all the results from previous modules and sub-modules and analyzes the possibility of different scenarios. Then it provides the driver with a visual display of the lane and traffic object detection results, as well as information about traffic conditions, using five properties:

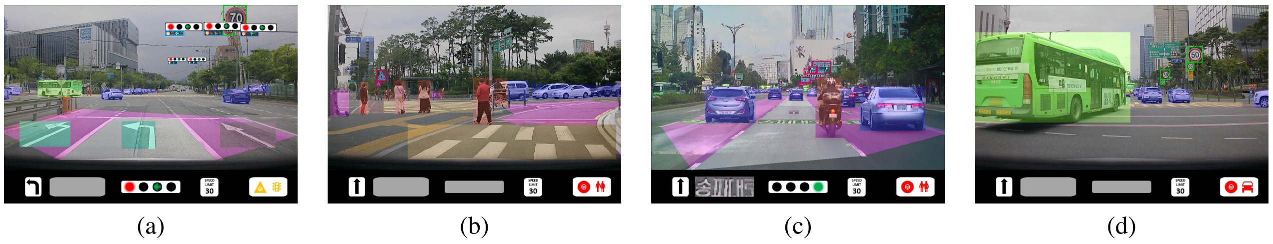

FIGURE 7. Example results of traffic scene analysis.

- Lane direction shows the direction of the current lane, which can be either straight by default or indicated by an arrow road marking.
- Speed limit shows the maximum speed limit for the current lane, which can be either 60 km/h by default or specified by a speed limit sign or number road marking.
- Traffic light indicates the state of a traffic light detected in the camera's view.
- Character road marking displays the guidance obtained from the character road markings.
- Driving condition informs the user about the current driving condition as analyzed by the system, which can be normal, warning, or danger. Normal indicates safe driving condition. The warning state means that the driver should be alert to possible obstacles on the road. The danger state indicates a high risk of collision with obstacles in close proximity in the ego-lane.

Fig. 2 shows the ADAS interface for displaying the all of that information.

The module determines the driving condition by checking different scenarios:

- Pedestrian check: In Korean traffic, pedestrians and bicycles often have similar moving paths, unlike motorcycles and other vehicles. It is important to distinguish pedestrians and bicycles from other vehicles due to their higher priority. The module checks two scenarios for these classes.
  The first scenario determines whether any pedestrian or bicycle is in the upcoming crosswalk or within the ego-lane region in front of the current vehicle:
  $$\begin{aligned}&\text{pedestrian-check}_{\text{danger}}\\&\quad= \text{IoU}(\text{bbox}_{\text{pedestrian}}, \text{bbox}_{\text{crosswalk}})\\&\quad\cup \text{IoU}(\text{bbox}_{\text{bicycle}}, \text{bbox}_{\text{crosswalk}})\\&\quad\cup \text{IoU}(\text{bbox}_{\text{pedestrian}}, \text{region}_{\text{ego-lane}})\\&\quad\cup \text{IoU}(\text{bbox}_{\text{bicycle}}, \text{region}_{\text{ego-lane}})\\&\quad> \epsilon_{\text{pedestrian-danger}}. \qquad (24)\end{aligned}$$
  The second scenario determines whether any pedestrian or bicycle is within other lane regions:
  $$\begin{aligned}&\text{pedestrian-check}_{\text{warning}}\\&\quad= \text{IoU}(\text{bbox}_{\text{pedestrian}}, \text{region}_{\text{all-lane}})\\&\quad\cup \text{IoU}(\text{bbox}_{\text{bicycle}}, \text{region}_{\text{all-lane}})\\&\quad> \epsilon_{\text{pedestrian-warning}} \qquad (25)\end{aligned}$$
  where IoU(,) is the intersection-over-union score between two polygons, $\epsilon_{\text{pedestrian-danger}}$ and $\epsilon_{\text{pedestrian-warning}}$ are the decision thresholds.
- Vehicle check: This module also checks two scenarios: The third scenario determines whether any other vehicle is within the lower one-third of the traffic frame, which indicates close proximity to the current vehicle:
  $$\begin{aligned}\text{bbox}_{\text{vehicle}} &= \text{bbox}_{\text{motorcycle}} \cup \text{bbox}_{\text{car}}\\&\quad\cup \text{bbox}_{\text{bus}} \cup \text{bbox}_{\text{truck}} \qquad (26)\\\text{bbox}_{\text{vehicle},i} &= (x_{i,\min}, y_{i,\min}, x_{i,\max}, y_{i,\max}) \qquad (27)\end{aligned}$$
  $$\text{vehicle-check}_{\text{danger}} = y_{i,\max} > \frac{2H}{3}(1 + \epsilon_{\text{vehicle-danger}}). \qquad (28)$$
  The fourth scenario determines whether any other vehicle is within the lower one-half of the traffic frame, which indicates moderate proximity to the current vehicle:
  $$\text{vehicle-check}_{\text{warning}} = y_{i,\max} > \frac{H}{2}(1 + \epsilon_{\text{vehicle-warning}}) \qquad (29)$$
  where H is the height of the traffic frame, and $\epsilon_{\text{vehicle-danger}}$ and $\epsilon_{\text{vehicle-warning}}$ are the decision thresholds.

If any of those scenarios is true, the module will change the driving condition flag to the corresponding state. Fig. 7 presents some example outputs of the scene analysis module.

B. SYSTEM DESIGN

The proposed system is designed to run on a single Jetson AGX Xavier, while still achieving real-time performance. To do this, the multithreading framework leverages different computing components on the Jetson embedded system, specifically the GPU and two DLA cores for deep learning tasks and the multi-core CPU for non–deep learning tasks. The system has six main threads, as shown in Fig. 8, which are the (1) Control thread, (2) Signal thread, (3) Lane Detection thread, (4) Traffic Object Detection thread, (5) Speed Limit Recognition thread and (6) Scene Analysis thread.

1) CONTROL THREAD

The Control thread oversees the creation and operation of the other functional threads within the system. It also monitors

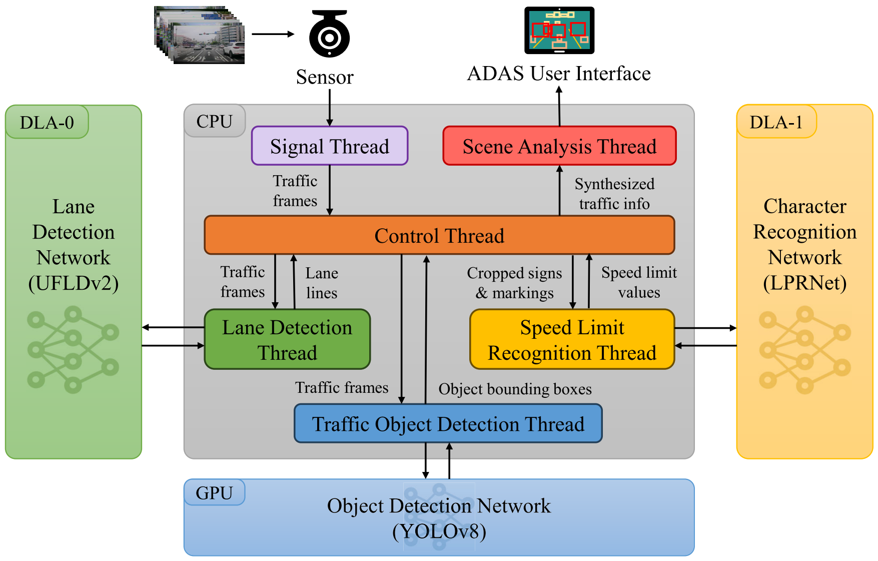

FIGURE 8. General system design on Jetson AGX Xavier. There are six main threads, with lane detection, traffic object detection and speed limit recognition threads are responsible for performing deep learning tasks on either the GPU or one of the two DLA cores.

TABLE 2. Device specifications.

	Jetson AGX Xavier DevKit	Jetson AGX Orin DevKit	Geforce Titan Xp
CPU	8x Carmel 2.26GHz	12x Cortex-A78AE 2.2GHz (60W) or 1.7GHz (30W)	Intel Core i7-7700 4x 3.60GHz
GPU	Volta GV10B 32GB, 512 CUDA cores, 64 Tensor cores, 1377MHz	Ampere GA10B 32GB, 2048 CUDA cores, 64 Tensor cores, 1300MHz (60W) or 625MHz (30W)	Pascal GP102 12GB (480.4GB/s), 3840 CUDA cores, 1582MHz
Accelerators	2x DLA v1 1.4GHz	2x DLA v2 1.6GHz (60W) or 1.4GHz (30W)	N/A
Memory	32GB (137GB/s) shared	32GB (204.8GB/s) shared	32GB
Power	10 | 15 | 30W	15 | 30 | 50 | 60W	250W
Theoretical Performance	FP32: 1.4 TFLOPs FP16: 16 TFLOPs INT8: 32 TOPs	FP32: 5.3 TFLOPs FP16: >85 Sparse TFOPs or >43 Dense TFOPs INT8: 275 Sparse TOPs or 138 Dense TOPs	FP32: 12.1 TFLOPs INT8: 48.4 TOPs
Software	JetPack 5.1.3, Ubuntu 20.04, CUDA 11.4, Python 3.8, Pytorch 2.1	JetPack 5.1.3, Ubuntu 20.04, CUDA 11.4, Python 3.8, Pytorch 2.1	Ubuntu 22.04, CUDA 11.8, Python 3.10, Pytorch 2.1.1

each thread's lifecycle and handle their termination and re-creation in response to error signals. In addition, the Control thread controls the flow of buffered data among the functional threads and ensures that they are in chronological order according to the frame index. This thread runs exclusively on the CPU.

2) SIGNAL THREAD

The Signal thread acts as the bridge between the system and the monocular camera system, establishing and maintaining a steady connection to capture traffic scenes. It receives traffic frames from the camera and stores them in a data buffer, which then serves as the input for the other functional threads. The Signal thread also attempts to solve by itself any connection or buffer issues that might arise before reporting an error signal to the Control thread. This thread operates only on the CPU as well.

3) LANE DETECTION THREAD

The Lane Detection thread is responsible for finding the boundary of the ego-lane and its left and right adjacent lanes that are within the camera's field of view. It receives the input traffic frame from the Control thread and feeds the cropped lower half to the network model that runs on the first DLA core of the edge device. Then, it examines the network's output to identify the lanes and returns that result back to the Control thread. Both the pre-processing and post-processing steps are executed on the CPU.

4) TRAFFIC OBJECT DETECTION THREAD

The Traffic Object Detection thread uses the GPU, the Jetson board's most powerful component, to execute a single YOLOv8 model for comprehensive traffic object detection, as explained in Section III-A2. This provides flexibility in the network model adjustment to optimize the system's most demanding task. The traffic frame, which is obtained from the data buffer and transferred here by the Control thread, is input to the network model. The output bounding boxes are then split into four groups, based on the four main categories of traffic objects. Each group is subjected to designated constraints to keep only the bounding boxes of interest within

TABLE 3. Model complexity and baseline inference throughput.

Model	Params (M)	FLOPs (G)	Baseline Throughput (FPS)
Lane Detection			
UFLDv2-res18-320x1600	206.30	37.40	28.65
UFLDv2-res34-320x1600	216.41	75.16	16.98
Object Detection			
YOLOv8s-640x640	11.16	28.60	38.98
YOLOv8m-640x640	25.89	78.90	16.25
YOLOv8s-1280x1280	11.16	114.40	11.85
YOLOv8l-640x640	43.67	165.20	9.24
YOLOv8m-1280x1280	25.89	315.80	4.48
Character Recognition			
LPRNet	0.45	0.29	127.29

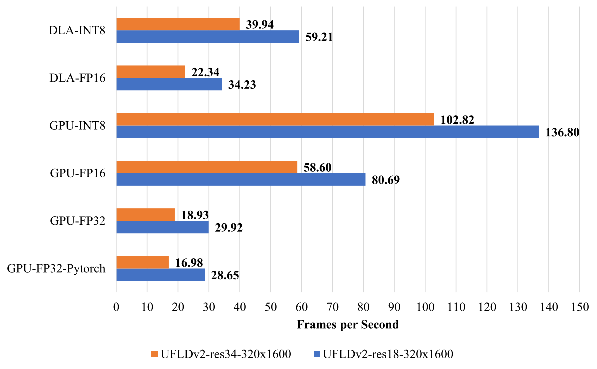

FIGURE 9. TensorRT inference throughput of UFLDv2 models with respect to GPU/DLA and quantization settings. -Pytorch suffix indicates baseline performance without using TensorRT framework.

specific regions of the traffic scene. After the post-processing steps on the CPU, the grouped bounding boxes are returned to the Control thread for further processing and analysis.

5) SPEED LIMIT RECOGNITION THREAD

Following the detection processes, the Control thread forwards the extracted ROIs of the speed limit traffic signs and number road markings to this Recognition thread to transform their content into text. The model is set up to run on the second DLA core of the embedded device, because it is not a heavily computational task. After that, the results are sent back to the Control thread before being transferred to the Scene Analysis thread with the other information to be analyzed.

6) SCENE ANALYSIS THREAD

The Scene Analysis thread runs only on the CPU, similar to the Signal thread and Control thread. It receives all the result data generated by the detection and recognition threads, as forwarded by the Control thread, and performs the scenario analysis for collision avoidance. This thread is also responsible for providing the synthesized information to the vehicle driver via a visual display.

7) OPTIMIZATION ON THE JETSON AGX XAVIER

The Jetson AGX Xavier was selected to be the main target embedded device for our system because it is the first device

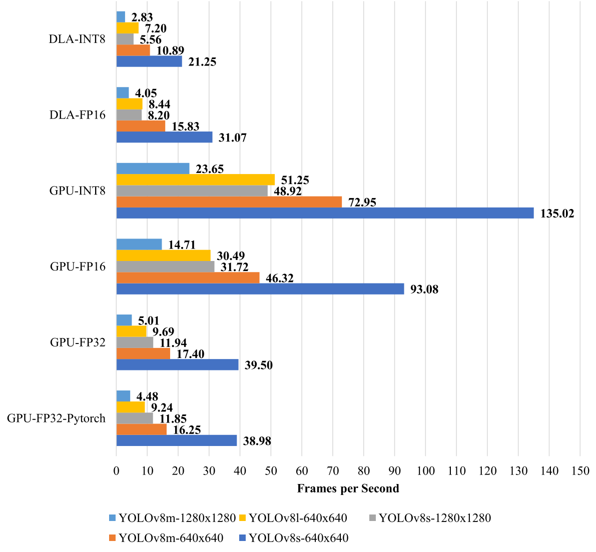

FIGURE 10. TensorRT inference throughput of YOLOv8 models with respect to GPU/DLA and quantization settings. -Pytorch suffix indicates baseline performance without using TensorRT framework.

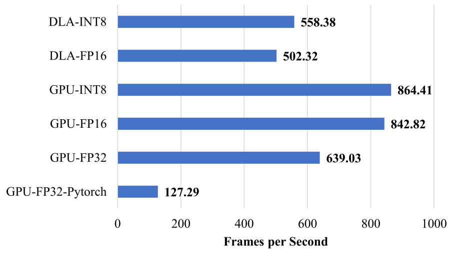

FIGURE 11. TensorRT inference throughput of LPRNet model with respect to GPU/DLA and quantization settings. -Pytorch suffix indicates baseline performance without using TensorRT framework.

to feature both tensor cores and two DLA cores, and it is capable of high performance at high power efficiency, relatively affordable, and well-supported by the Jetpack software development kit. A tensor core is a specialized processing unit in an NVIDIA GPU since the Volta architecture, and is designed to accelerate convolutions and matrix operations in deep learning and high-performance computing by using lower-precision formats [39]. The NVIDIA DLA is an open hardware architecture designed for deep learning inference with high power efficiency. It can work as an additional accelerator in parallel with the GPU. Table 2 shows the theoretical performance of the edge device. The utilization of the two DLA cores and the GPU, accelerated by the Tensor cores, and Nvidia's TensorRT optimization technology and the post-training quantization technique allow us to make use of all the processing power available on the hardware.

To achieve that goal, we conducted extensive benchmarks on the three deep learning tasks in the system, the UFLDv2 lane detection, YOLOv8 object detection, and LPRNet character recognition, to measure their inference throughput and find the best arrangement for the tasks on the device's

TABLE 4. Comparison of traffic object detection AP50 accuracy (%).

	YOLOv8m 640x640	**YOLOv8l 640x640**	**YOLOv8s 1280x1280**	**Scaled-YOLOv4 1280x1280** [5]	**DLT-Net** [34]	**YOLOP** [35]
Vehicle & pedestrian	89.8	94.2	92.2	84.1	65.2	70.0
Traffic light	67.1	73.5	88.0	83.6	62.7	64.6
Traffic sign	98.4	98.9	98.5	96.2	81.8	84.2
Road marking	89.3	93.7	91.6	79.7	50.6	51.6
All	85.4	89.7	91.6	83.1	59.8	62.1

TABLE 5. Comparison of scene analysis accuracy (%).

	UFLDv2 YOLOv8m-640 LPRNet	**UFLDv2 YOLOv8l-640 LPRNet**	**UFLDv2 YOLOv8s-1280 LPRNet**	**UFLD Scaled-YOLOv4 LPRNet** [5]
Lane direction	98.6	98.9	98.7	98.1
Traffic light	85.3	93.4	98.5	97.5
Speed limit	95.8	98.5	97.1	95.6
Driving condition	96.6	98.1	96.9	96.0
All	94.1	97.2	97.8	96.8

computing units. At first, we set the power profile to the highest setting at 30W, which we consider reasonable for ADAS, to enable full-capacity horsepower and disabled the dynamic voltage and frequency scaling governor to ensure the device always had that state of processing power. Then, with each task executed on either the GPU or a DLA core, we implemented a network inference pipeline based on the TensorRT framework for device-specific throughput and latency optimization. In addition, we applied post-training quantization to further improve network speed. For the experiments we used different network model variants with three precision modes: non-quantized single-precision floating-point (FP32), half-precision floating-point (FP16), and 8-bit fixed-point (INT8). On the DLA core, we could only use the latter two modes, as it does not support FP32 precision.

The baseline inference throughput with the number of parameters (Params) and floating-point operations per second (FLOPs) of the models are presented in Table 3. Params and FLOPs, which respectively indicate space and time complexities, do not precisely reflect a model's processing speed on a specific device. While complexity indicators can provide a rough idea of relative performance between different network variants or networks sharing similar architecture, their actual performance can only be accurately measured through benchmarking. The optimization level of both hardware and the framework significantly impacts deep learning operations and overall network performance.

The benchmark results in Figs. 9, 10, and 11 show that the TensorRT only marginally improved processing speed on the power-constrained device. The only exception was the lightweight LPRNet model, which achieved a 402% boost. The throughput of the UFLDv2, YOLOv8, and LPRNet models was increased significantly by quantizing to the FP16 precision (up to 210%, 215%, and 32%, respectively) and by quantizing to the INT8 precision (up to 75%, 68%, and 3%, respectively). This pattern did not always apply to the DLA core, where performance of the simpler UFLDv2 and LPRNet models improved by up to 79% and 11%, but the that of the YOLOv8 models worsened by up to 32% when comparing INT8 to FP16 throughput. Moreover, FP16 throughput on the DLA core was substantially slower (up to 3.9 times) than on the GPU. This is, to some extent, consistent with the theoretical performance provided by Nvidia (Table 2). Nonetheless, several model variants on the DLA core still achieved real-time processing speed.

From the inference throughput comparison between tasks, we determined that object detection was the most computing-intensive, and that lane detection and character recognition tasks could run relatively well on the DLA core. Based on this, we arranged the tasks as described above in previous subsections (III-B1 to III-B6). From the components that individually satisfy real-time performance, we set up seven configurations for whole-system performance evaluation including one FP16 system and six INT8 systems. The FP16 system consisted of the UFLDv2-res18, YOLOv8m-640 $\times$ 640, and LPRNet models, all quantized to FP16 precision. The six INT8 systems used different combinations of INT8-quantized UFLDv2-res18 or UFLDv2-res34 for lane detection models, YOLOv8m-640 $\times$ 640, YOLOv8l-640 $\times$ 640 or YOLOv8s-1280 $\times$ 1280 for object detection, and LPRNet for character recognition.

IV. EXPERIMENTS

A. EXPERIMENT SETUP

The KATECH Korean dashcam traffic dataset [15], [16] was used for training and validating the performance of the proposed ADAS. The dataset consists of more than 160,000 traffic frames at a resolution of 1280 $\times$ 720, 1280 $\times$ 672, or 1920 $\times$ 1080. The frames are from 24 traffic sequences captured in different lighting and weather conditions: daytime, dawn, nighttime, sunny, overcast, and rainy. For the detection task, we split the dataset into training and validation sets with an 80:20 ratio.

We measured the accuracy of the traffic object detection task on the validation set using AP50 following the COCO detection evaluation metrics, which is the average precision at an IoU threshold of 50%. For scene analysis for collision avoidance task, we used 10,000 images with added annotations about the traffic condition properties, including six lane directions, five traffic light states, speed limits from 30 to 150 km/h, and three driving conditions [5]. We used the

TABLE 6. Comparison of whole system throughput (FPS) on different hardware.

System	Lane Detection Model	Traffic Object Detection Model	Character Recognition Model	Precision Format	Jetson AGX Xavier	Jetson AGX Orin (30W)	Jetson AGX Orin (60W)	Geforce Titan Xp
FP16	UFLDv2-res18-320x1600	YOLOv8m-640x640	LPRNet-24x94	FP16	25.04	10.69	12.33	61.56
INT8 #1	UFLDv2-res18-320x1600	YOLOv8m-640x640	LPRNet-24x94	INT8	43.67	37.42	131.36	111.59
INT8 #2	UFLDv2-res18-320x1600	YOLOv8l-640x640	LPRNet-24x94	INT8	42.74	29.59	98.87	76.38
INT8 #3	UFLDv2-res18-320x1600	YOLOv8s-1280x1280	LPRNet-24x94	INT8	38.51	29.31	90.07	76.00
INT8 #4	UFLDv2-res34-320x1600	YOLOv8m-640x640	LPRNet-24x94	INT8	31.18	38.72	89.04	106.38
INT8 #5	UFLDv2-res34-320x1600	YOLOv8l-640x640	LPRNet-24x94	INT8	30.79	28.54	103.76	77.79
INT8 #6	UFLDv2-res34-320x1600	YOLOv8s-1280x1280	LPRNet-24x94	INT8	30.74	24.47	91.10	68.24

accuracy metric to measure the performance of the proposed ADAS in analyzing traffic conditions in this subset:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{30}$$

where a prediction is considered correct if it accurately identifies lane direction, traffic light, speed limit or driving condition. The overall accuracy is the average of the accuracies across these four properties. The results of the task should reflect the overall performance of the entire driving assistance system. We compared both results to the system in [5] because that system and ours have a similar collection of tasks.

Processing time performance is the most important evaluation criterion, as the focus of this study. We measured the whole system throughput of the aforementioned seven configurations on three different platforms: Jetson AGX Xavier, Jetson AGX Orin at the same 30W power profile and at its full 60W power profile, and a workstation system with a single Nvidia Geforce Titan Xp GPU. Table 2 presents the detailed system specifications of the platforms.

To ensure reliability, the throughput was measured over a 2000-frame period, excluding the initial warm-up stage. Furthermore, each experiment was conducted three times, and the measurements were averaged to obtain the final result.

B. EXPERIMENTAL RESULTS AND DISCUSSION

Table 4 shows the traffic object detection performance of the proposed system's configurations. All three YOLOv8 models performed better than DLT-Net [34] and YOLOP [35] in every category, and better than the model in [5] in most categories. The exception was traffic light detection, where YOLOv8m-640 $\times$ 640 and YOLOv8l-640 $\times$ 640 had lower accuracy than Scaled-YOLOv4-1280 $\times$ 1280 in [5]. This can be attributed to several factors. First, the model in [5] was specialized to traffic light detection only, instead of being trained to detect all traffic object classes. Second, traffic lights are usually smaller than other traffic objects, as seen in Fig. 7a. Third, the selected YOLOv8m and YOLOv8l models were trained with a smaller input size than the Scaled-YOLOv4 model, which affected their ability to detect small objects. The YOLOv8s model, which was trained with 1280$\times$ 1280 input size, did not have this problem and outperformed the Scaled-YOLOv4 model.

The scene analysis performance of the proposed system is shown in Table 5, and it is consistent with the traffic object detection accuracy results. The three testing configurations of the proposed system performed better than [5] in lane direction, speed limit, and driving condition analysis. However, traffic light analysis depended on traffic light detection results, so the configurations using YOLOv8m-640 $\times$ 640 and YOLOv8l-640 $\times$ 640 for object detection also had lower accuracy, while the configuration with YOLOv8s-1280 $\times$ 1280 had the highest accuracy.

Table 6 lists the throughput, of the whole system for each of seven testing configurations on different hardware. Several observations could be made regarding their performance on the Jetson AGX Xavier. The FP16 system barely reached 25 FPS, while the INT8-quantized configurations exceeded 30 FPS. The system was constrained by the performance of the lane detection model on the DLA core. Switching from the res34 variant to the simpler res18 notably improved overall performance, which was then limited only by the performance of the object detection model. Next, system throughput was much lower than individual module throughput for several reasons. First, the DLA has a limited number of supported layers, and some of those layers have restricted parameters. The unsupported layers fall back to be processed by the GPU, adding overhead to the computation. Second, the GPU and the two DLA cores share the same system memory pool. If the parallel workload is too bandwidth-intensive, both will be slowed down by memory access. Despite this, the proposed system comfortably satisfied the real-time processing requirement, proving its viability for real-world applications.

Next, we compared system performance on two Jetson boards at the 30W power profile. The AGX Orin performed worse than the AGX Xavier. This was because at 30W, the frequency of both the GPU and the DLA cores decreased by 50% and 12.5%, respectively. This caused the system to be bottlenecked by the object detection model on the GPU most of the time. Changing the lane detection variants had little effect on performance. The seventh configuration, with the UFLDv2-res34 and YOLOv8s-1280 $\times$ 1280, was too demanding and caused performance degradation. Furthermore, the second version of the DLA core on the AGX Orin was better optimized for INT8 using a FP16 performance trade-off, which resulted in 24% higher FPS in the first INT8 configuration and 57% lower FPS in the FP16 configuration. This showed that the Jetson AGX Xavier was more suitable for ADAS implementation when using the same power profile.

System performance improved greatly when we increased the AGX Orin's power to its highest 60W, with FPS increasing by 131% to 237%. This was due to the new Ampere GPU and DLAv2 optimized for INT8. At the expense of higher installation cost and power consumption, this provides the option to scale up the modular system design by employing more complex, higher performance network models, adding more input sensors or more advanced functionalities to fully exploit the processing capacity. However, the FP16 system's FPS only improved by 15% compared to its performance at 30W, and remained 51% lower than the AGX Xavier.

The Titan Xp workstation is much more powerful than the Jetson AGX Xavier, as it can run all ADAS functions on the GPU with up to 3.4-fold higher FPS. However, installing a desktop system in a vehicle for driving assistance is not practical, as the GPU alone already consumes 250W of power. Moreover, the Jetson AGX Orin, which is five years newer, has higher performance than the Titan Xp in most testing configurations with only one-fourth of the power consumption. However, the Jetson AGX Xavier is still a versatile option considering its affordability, efficiency and capability to deliver real-time ADAS performance.

Finally, compared to the distributed computing system in [5], which shares similar set of features and achieves 19.61 FPS, our solution significantly increases processing speed while maintaining comparable performance. Notably, our implementation operates substantially more efficient on a single Jetson AGX Xavier, in contrast to a cluster of five separate edge devices. The cloud-based offloading systems in [12], [13], and [14] have a different feature set, but may achieve higher task-specific throughput due to their powerful remote cloud servers. However, their overall performance suffers from high edge-cloud data transfer latency (at least 2 seconds/image), failing to meet the real-time requirement. Lastly, MTSan, which has a multi-task network running entirely on the Jetson AGX Xavier's GPU without any optimization, achieves inferior performance at 10 FPS. This demonstrates the effectiveness of our system design and optimization.

V. CONCLUSION

This study presents a real-time driving assistance system implemented entirely on an edge device that performs lane detection, traffic object detection and recognition, and scene analysis for collision avoidance. We leverage the full processing power of the NVIDIA Jetson AGX Xavier, particularly the DLA cores; and we optimize the system with multithreaded processing, the NVIDIA TensorRT inference framework, and post-training quantization to satisfy the real-time requirement. Based on a modular architecture, the system is scalable, upgradable and adaptable to different traffic regulations and conditions. The experimental results demonstrate the system's effectiveness, efficiency, and capability for real-world deployment. This research offers valuable insights into the potential and advantages of Nvidia DLA on the Jetson platform and NPUs in general in developing highly efficient deep learning solutions. As part of our future work, we plan to explore advanced optimization approaches, e.g. those proposed in [43] and [44], to further improve the computation time and overall performance of our framework.

REFERENCES

[1] K. Muhammad, A. Ullah, J. Lloret, J. D. Ser, and V. H. C. de Albuquerque, "Deep learning for safe autonomous driving: Current challenges and future directions," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4316–4336, Jul. 2021.

[2] J. Jhung, H. Suk, H. Park, and S. Kim, "Hardware accelerators for autonomous vehicles," in *Artificial Intelligence and Hardware Accelerators*. Cham, Switzerland: Springer, 2023, pp. 269–317.

[3] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti, and G. S. Sachdev, "Compute solution for teslas full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Feb. 2020.

[4] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. Patterson, "TPU V4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," 2023, *arXiv:2304.01433*.

[5] D. N. Tran, L. H. Pham, H.-H. Nguyen, T. H. Tran, H.-J. Jeon, and J. W. Jeon, "Universal detection-based driving assistance using a mono camera with Jetson devices," *IEEE Access*, vol. 10, pp. 59400–59412, 2022.

[6] P. Azevedo and V. Santos, "Comparative analysis of multiple YOLO-based target detectors and trackers for ADAS in edge devices," *Robot. Auto. Syst.*, vol. 171, Jan. 2024, Art. no. 104558.

[7] X. Yu, L. Qin, X. Chen, L. Wu, and B. Zhang, "Research on optimization of neural network model deployment for edge devices," in *Proc. 4th Int. Conf. Comput. Eng. Intell. Control (ICCEIC)*, Oct. 2023, pp. 1–12.

[8] K. Podbucki, J. Suder, T. Marciniak, and A. Dabrowski, "Evaluation of embedded devices for Real- time video lane detection," in *Proc. 29th Int. Conf. Mixed Design Integr. Circuits Syst. (MIXDES)*, Jun. 2022, pp. 187–191.

[9] S. Zheng, Y. Xie, M. Li, C. Xie, and W. Li, "A novel strategy for global lane detection based on key-point regression and multi-scale feature fusion," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 23244–23253, Dec. 2022.

[10] G. Tatar and S. Bayar, "Real-time multi-task ADAS implementation on reconfigurable heterogeneous MPSoC architecture," *IEEE Access*, vol. 11, pp. 80741–80760, 2023.

[11] S. Miraliev, S. Abdigapporov, J. Alikhanov, V. Kakani, and H. Kim, "Edge device deployment of multi-tasking network for self-driving operations," 2022, *arXiv:2210.04735*.

[12] J. Tang, S. Liu, L. Liu, B. Yu, and W. Shi, "LoPECS: A low-power edge computing system for real-time autonomous driving services," *IEEE Access*, vol. 8, pp. 30467–30479, 2020.

[13] M. J. Khan, M. A. Khan, S. Turaev, S. Malik, H. El-Sayed, and F. Ullah, "A vehicle-edge-cloud framework for computational analysis of a fine-tuned deep learning model," *Sensors*, vol. 24, no. 7, pp. 1–9, 2024.

[14] Y. Yuan, S. Gao, Z. Zhang, W. Wang, Z. Xu, and Z. Liu, "Edge-cloud collaborative UAV object detection: Edge-embedded lightweight algorithm design and task offloading using fuzzy neural network," *IEEE Trans. Cloud Comput.*, vol. 12, no. 1, pp. 306–318, Jan. 2024.

[15] *Korea Automotive Technology Institute*. Accessed: Jan. 25, 2024. [Online]. Available: http://www.katech.re.kr/eng

[16] D. N. Tran, H.-H. Nguyen, L. H. Pham, and J. W. Jeon, "Object detection with deep learning on drive PX2," in *Proc. IEEE Int. Conf. Consum. Electron.*, Nov. 2020, pp. 1–4.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.

[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 936–944.

[19] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into high quality object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6154–6162.
[20] A. Pramanik, S. K. Pal, J. Maiti, and P. Mitra, "Granulated RCNN and multi-class deep SORT for multi-object detection and tracking," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 6, no. 1, pp. 171–181, Feb. 2022.
[21] C.-Y. Wang, A. Bochkovskiy, and H. M. Liao, "Scaled-YOLOV4: Scaling cross stage partial network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Nashville, TN, USA, Jun. 2021, pp. 13024–13033.
[22] C.-Y. Wang, I.-H. Yeh, and H. Liao, "You only learn one representation: Unified network for multiple tasks," *J. Inf. Sci. Eng.*, vol. 39, no. 3, pp. 691–709, 2023.
[23] C.-Y. Wang, A. Bochkovskiy, and H.-Y.-M. Liao, "YOLOV7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Vancouver, BC, Canada, Jun. 2023, pp. 7464–7475.
[24] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021, *arXiv:2107.08430*.
[25] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "YOLOV6: A single-stage object detection framework for industrial applications," 2022, *arXiv:2209.02976*.
[26] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOV1 to YOLOV8 and YOLO-NAS," *Mach. Learn. Knowl. Extraction*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023.
[27] T. Zheng, H. Fang, Y. Zhang, W. Tang, Z. Yang, H. Liu, and D. Cai, "RESA: Recurrent feature-shift aggregator for lane detection," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 4, May 2021, pp. 3547–3554.
[28] H. Abualsaud, S. Liu, D. B. Lu, K. Situ, A. Rangesh, and M. M. Trivedi, "LaneAF: Robust multi-lane detection with affinity fields," *IEEE Robotics Automation Letters*, vol. 6, no. 4, pp. 7477–7484, Oct. 2021.
[29] L. Tabelini, R. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Keep your eyes on the lane: Real-time attention-guided lane detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Nashville, TN, USA, Jun. 2021, pp. 294–302.
[30] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *Computer Vision*. Cham, Switzerland: Springer, 2020, pp. 276–291.
[31] Z. Qin, P. Zhang, and X. Li, "Ultra fast deep lane detection with hybrid anchor driven ordinal classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, pp. 2555–2568, 2022.
[32] L. Tabelini, R. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "PolyLaneNet: Lane estimation via deep polynomial regression," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 6150–6156.
[33] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, and R. Urtasun, "MultiNet: Real-time joint semantic reasoning for autonomous driving," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 1013–1020.
[34] Y. Qian, J. M. Dolan, and M. Yang, "DLT-Net: Joint detection of drivable areas, lane lines, and traffic objects," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 11, pp. 4670–4679, Nov. 2020.
[35] D. Wu, M.-W. Liao, W.-T. Zhang, X.-G. Wang, X. Bai, W.-Q. Cheng, and W.-Y. Liu, "YOLOP: You only look once for panoptic driving perception," *Mach. Intell. Res.*, vol. 19, no. 6, pp. 550–562, Dec. 2022.
[36] C. Han, Q. Zhao, S. Zhang, Y. Chen, Z. Zhang, and J. Yuan, "YOLOPV2: Better, faster, stronger for panoptic driving perception," 2022, *arXiv:2208.11434*.
[37] D. Vu, B. Ngo, and H. Phan, "HybridNets: End-to-end perception network," 2022, *arXiv:2203.09035*.
[38] C.-Y. Lai, B.-X. Wu, V. M. Shivanna, and J.-I. Guo, "MTSAN: Multi-task semantic attention network for ADAS applications," *IEEE Access*, vol. 9, pp. 50700–50714, 2021.
[39] A. Mishra, P. Yadav, and S. Kim, *Artificial Intelligence Accelerators, in Artificial Intelligence and Hardware Accelerators*. Cham, Switzerland: Springer, 2023, p. 152.
[40] A. Avramovic, D. Sluga, D. Tabernik, D. Skocaj, V. Stojnic, and N. Ilc, "Neural-network-based traffic sign detection and recognition in high-definition images using region focusing and parallelization," *IEEE Access*, vol. 8, pp. 189855–189868, 2020.
[41] S. Zherzdev and A. Gruzdev, "LPRNet: License plate recognition via deep neural networks," 2018, *arXiv:1806.10447*.
[42] Jetson Benchmarks. *NVIDIA Developer*. Accessed: Jan. 25, 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-benchmarks
[43] M. Mollajafari and H. S. Shahhoseini, "A repair-less genetic algorithm for scheduling tasks onto dynamically reconfigurable hardware," *Int. Rev. Comput. Softw.*, vol. 6, no. 3, 2011, Art. no. 206212.
[44] M. Negahban, M. V. Ardalani, M. Mollajafari, E. Akbari, M. Talebi, and E. Pouresmaeil, "A novel control strategy based on an adaptive fuzzy model predictive control for frequency regulation of a microgrid with uncertain and time-varying parameters," *IEEE Access*, vol. 10, pp. 57514–57524, 2022.

HUY-HUNG NGUYEN received the B.S. degree in computer science and the M.E. degree in information technology management from International University—Vietnam National University, Vietnam, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Sungkyunkwan University, Suwon, South Korea. His research interests include computer vision, image processing, and deep learning.

DUONG NGUYEN-NGOC TRAN (Member, IEEE) received the B.S. degree in computer science and the M.S. degree in information technology management from International University, Ho Chi Minh City, Vietnam, in 2014 and 2018, respectively, and the Ph.D. degree in electrical and computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2023. He is currently a Postdoctoral Researcher with the Automation Laboratory, Sungkyunkwan University. His current research interests include computer vision, image processing, and deep learning.

LONG HOANG PHAM (Member, IEEE) received the B.S. degree in computer science and the M.S. degree in information technology management from International University, Ho Chi Minh City, Vietnam, in 2013 and 2017, respectively, and the Ph.D. degree in electrical and computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2021. He is currently a Postdoctoral Researcher with the Automation Laboratory, Sungkyunkwan University. His current research interests include computer vision, image processing, and deep learning.

JAE WOOK JEON (Senior Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1984 and 1986, respectively, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1990. From 1990 to 1994, he was a Senior Researcher with Samsung Electronics, Suwon, South Korea. Since 1994, he has been with Sungkyunkwan University, Suwon, where he was an Assistant Professor with the School of Electrical and Computer Engineering and is currently a Professor with the School of Information and Communication Engineering. His current research interests include robotics, embedded systems, and factory automation.