THESIS

OPTIMIZING MACHINE LEARNING MODELS FOR AUTONOMOUS VEHICLES

Submitted by

Abhishek Balasubramaniam

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2025

Master's Committee:

Advisor: Sudeep Pasricha

Edwin Chong,
Louis-Noël Pouchet

ABSTRACT

OPTIMIZING MACHINE LEARNING MODELS FOR AUTONOMOUS VEHICLES

Object detectors (ODs) stand as a cornerstone of modern computer vision tasks, increasingly essential in a wide array of consumer applications. Its utility spans enhancing surveillance and security systems, enabling mobile text recognition for digital document accessibility, and facilitating the diagnosis of diseases through advanced imaging techniques like MRI and CT scans. This multifaceted technology is pivotal across various domains, with one of its most critical applications being autonomous driving. Autonomous vehicles (AVs) rely extensively on their ability to perceive and interpret their surroundings, a capability fundamental to ensuring safe and reliable driving performance. Sophisticated perception systems in these vehicles utilize state-of-the-art object detection algorithms, both 2D and 3D, to accurately identify and localize various objects within the vehicle's operational vicinity. 2D ODs are designed to detect and localize objects in images or video frames, providing information in the form of bounding boxes on a 2-dimensional plane. They are less complex and computationally demanding compared to 3D detectors and are commonly used in applications like image recognition, face detection, and pedestrian detection in surveillance systems. Models such as YOLO, SSD, and Faster R-CNN are widely used examples of 2D ODs. Conversely, 3D ODs incorporate depth information to detect and localize objects in a three-dimensional space, utilizing data from 3D sensors like LiDAR, stereo cameras, or depth cameras. These detectors are essential for applications requiring a precise understanding of the environment, such as autonomous driving, robotics, and augmented reality.

Popular models include PointNet, VoxelNet, and Frustum PointNet. The data provided by these ODs, especially when combining 2D and 3D capabilities, is indispensable for informing crucial driving decisions and enabling the vehicle to navigate complex environments with enhanced safety and efficiency. However, these advanced ODs come with high memory and computational overheads, which pose significant challenges.

To address this challenge, ongoing research and development efforts are dedicated to optimizing these models. The primary goal is to reduce their memory footprint and computational requirements while maintaining or even improving their performance. This ensures that these sophisticated algorithms can be efficiently deployed on resource-constrained embedded platforms, often used in AVs, without compromising their effectiveness. Such advancements are pivotal in maintaining the efficiency and reliability of AVs, further solidifying the indispensable role of ODs in modern technology. This thesis introduces two novel OD optimization algorithms, which can reduce model footprint and computation cost while decreasing the inference time of the model. The first contribution, *R-TOSS*, is a novel semi-structured pruning framework for 2D ODs. *R-TOSS* outperforms various state-of-the-art model optimization techniques while also improving performance on embedded resource-constrained platforms. For accelerating 3D ODs, we propose *UPAQ*, which uses a combination of pruning and quantization to improve model accuracy and reduce model footprint. We also showcase how *UPAQ* outperforms other state-of-the-art models in terms of performance.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this thesis possible.

I would like to express my sincere gratitude to my advisor, Dr. Sudeep Pasricha, who has guided me through every step of my research during my Master's Thesis. His professional expertise and constant encouragement have been instrumental in shaping my academic and personal growth. I am deeply grateful for the opportunities and challenges that he has provided me with, which have motivated me to embark on new academic and personal endeavors. Throughout my research journey, he has been a constant source of inspiration, providing me with invaluable advice and feedback that has allowed me to navigate the complexities of graduate school life with confidence. I had the indispensable opportunity to learn a great deal from him about Computer Architecture, Machine Learning, Silicon Photonics, and Embedded Systems. Moreover, I am extremely thankful for all his patience and mentorship which have enabled me to develop new skills and explore challenging problems in these domains.

I would like to take this opportunity to thank the respected members of my Master's Thesis committee, Dr. Edwin Chong, and Dr. Louis-Noël Pouchet for their valuable time and feedback.

Furthermore, I would like to thank my research partner, Febin Sunny, for his collaboration, expertise, and support throughout the course of our research projects. I am especially grateful for his willingness to engage in intellectually stimulating conversations that have pushed me to think more deeply about our research questions and have boosted my morale when faced with complex problems. This list cannot be complete without mentioning the company and the help from my current lab mates in Dr. Pasricha's EPIC lab as well.

I am grateful for my wonderful family – my grandparents Saroja and Arumugam, my father Balasubramaniam Arumugam, my mother Pushpavalli Balasubramaniam, my sibling Lokesh Balasubramaniam, my uncle Kaliaperumal Arumugam, and my aunt Uma Kannaiyan – who have supported me throughout my years. As exemplary role models, they ignited my interest in the research field and provided me with the encouragement I need to pursue my graduate studies.

TABLE OF CONTENTS

LIST OF RESEARCH PUBLICATIONS

YET TO BE PUBLISHED:

1. Balasubramaniam, and S. Pasricha. “Object detection in autonomous vehicles: Status and open challenges.” *arXiv preprint arXiv:2201.07706* (2022).

CONFERENCE PUBLICATIONS:

1. A. Balasubramaniam, F. Sunny and S. Pasricha, “*R-TOSS*: A Framework for Real-Time Object Detection using Semi-Structured Pruning,” *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-6, doi: 10.1109/DAC56929.2023.10247917.
2. A. Balasubramaniam, F. Sunny and S. Pasricha. (2025) “*UPAQ*: A Framework for Real-Time and Energy-Efficient 3D Object Detection in Autonomous Vehicles”, *2025 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Lyon, France, 2025

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1 INTRODUCTION

This chapter presents an overview of Object Detectors (ODs) and outlines the challenges faced when deploying these ODs on resource-constraint embedded platforms. We discuss the fundamental limitations in optimizing these ODs using various optimization techniques such as pruning, quantization, and knowledge distillation. This chapter also presents a general overview of the contributions of this thesis.

1.1 LATEST TRENDS IN AUTONOMOUS VEHICLES

Autonomous vehicles (AVs) are revolutionizing transportation by reducing human error, improving traffic flow, and optimizing routes, resulting in faster and safer travel. These advancements are powered by cutting-edge sensor technology, artificial intelligence (AI), and connectivity. AVs utilize an array of sensors, including LiDAR, radar, cameras, and ultrasonic sensors, to generate a detailed view of their surroundings and detect obstacles, pedestrians, and other vehicles [1]. AI algorithms process this sensor data using machine learning models, predict potential scenarios, and make real-time decisions, continuously improving from the data collected [2]. Vehicle-to-Everything (V2X) communication connects AVs to the internet and other vehicles, enabling them to share information about traffic conditions, road hazards, and optimal routes, thus enhancing overall traffic flow and safety [3]. Advanced Driver Assistance Systems (ADAS), such as adaptive cruise control, lane-keeping assist, and automatic emergency braking, are integrated into AVs to bolster safety and driving comfort.

The California Department of Motor Vehicles reported that AVs had a crash rate of 26.3 per million vehicle miles traveled in 2022, compared to 1.9 for all motor vehicles in the U.S. Although AVs had a higher crash rate, many incidents were minor and often caused by other road users [4]. Research indicates that the widespread adoption of AVs could reduce traffic congestion by up to 30%, thanks to optimized driving patterns and enhanced traffic management systems [5]. Moreover, AVs could significantly decrease greenhouse gas emissions by up to 60%, due to more efficient driving practices, reduced idling, and optimized route planning [6]. The autonomous vehicle market is projected to generate between 300 and 400 billion dollars in revenue by 2035, fueled by technological advancements and growing consumer adoption [7].

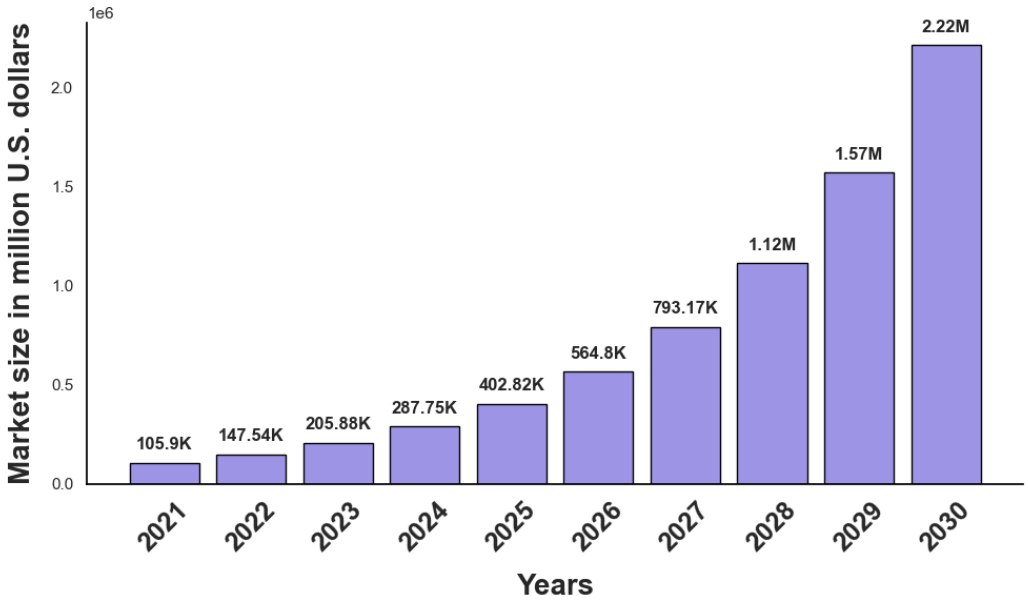


Figure 1 Trend in market size increase over years [4]

The global market for Level 1 (L1) and Level 2 (L2) AVs reached approximately USD 106 billion in 2021 and is expected to exceed USD 2.2 trillion by 2030, with a compound annual growth rate (CAGR) of 35.6% from 2021 to 2030 (Figure 1) [4]. This remarkable growth is driven by the demand for safe, efficient, and convenient driving experiences. AVs, equipped with advanced

safety features like collision avoidance systems, lane departure warnings, and automatic emergency braking, appeal to consumers who prioritize safety and convenience [5]. Additionally, rising disposable income in emerging economies is boosting the AVs market, as financially stable individuals are more likely to invest in advanced technologies, including AVs. Strict safety regulations worldwide are compelling manufacturers to adopt autonomous technologies, leading to increased research and development and resulting in more advanced and reliable autonomous driving systems [6].

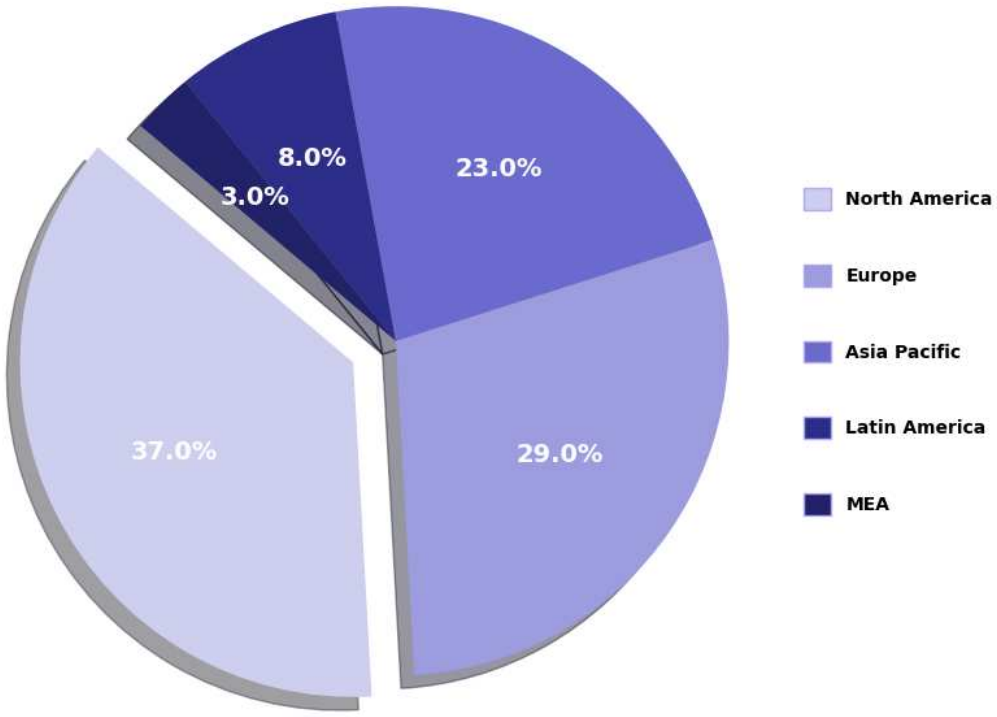


Figure 2 Market Share of Autonomous Vehicles by Region [7]

Figure 2 shows that North America region is projected to hold the largest market share by 2030, followed by Europe and Asia Pacific. Rapid urbanization and technological advancements in countries like China, Japan, and South Korea are driving the adoption of AVs in the Asia Pacific

region. Europe and North America are also experiencing significant growth due to supportive government policies and increasing investments [7].

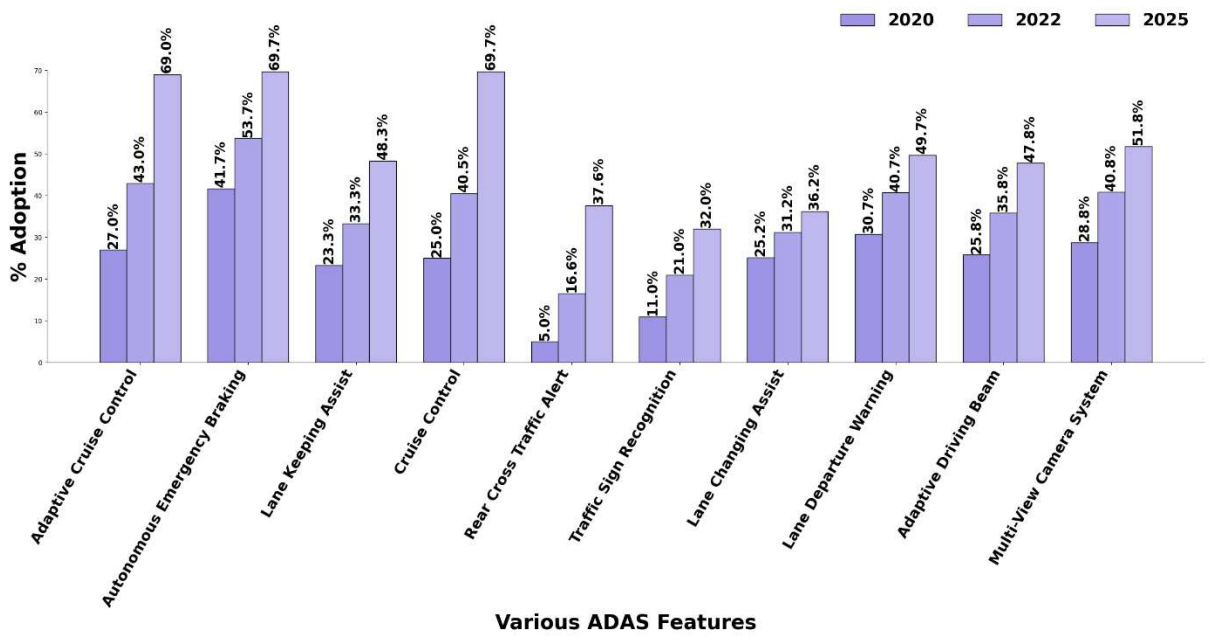


Figure 3 Growth of ADAS Features over the years [8]

The steady increase in the adoption of Advanced Driver Assistance Systems (ADAS) features, as shown in Figure 3, underscores the growing reliance on intelligent perception systems in modern vehicles. Object detection plays a crucial role in enhancing the effectiveness of these systems, ensuring safety, efficiency, and overall vehicle autonomy.

ODs are fundamental to ADAS functionalities such as Autonomous Emergency Braking, Lane Keeping Assist, Adaptive Cruise Control, and Traffic Sign Recognition, all of which have seen notable adoption growth between 2020 and 2025 [8]. These systems depend on accurate and real-time object recognition to identify vehicles, pedestrians, road signs, and other environmental elements. Improvements in object detection algorithms directly translate to higher accuracy, reducing false positives and negatives, which is critical for safety-critical applications like Rear Cross Traffic Alert and Lane Departure Warning.

For AVs, the role of object detection extends beyond ADAS, forming the backbone of perception systems. Advanced sensors, including LiDAR, radar, and cameras, coupled with deep learning-based ODs, enable self-driving cars to interpret their surroundings with high precision. Enhancing these detection models improves decision-making, navigation in complex environments, and overall vehicle safety.

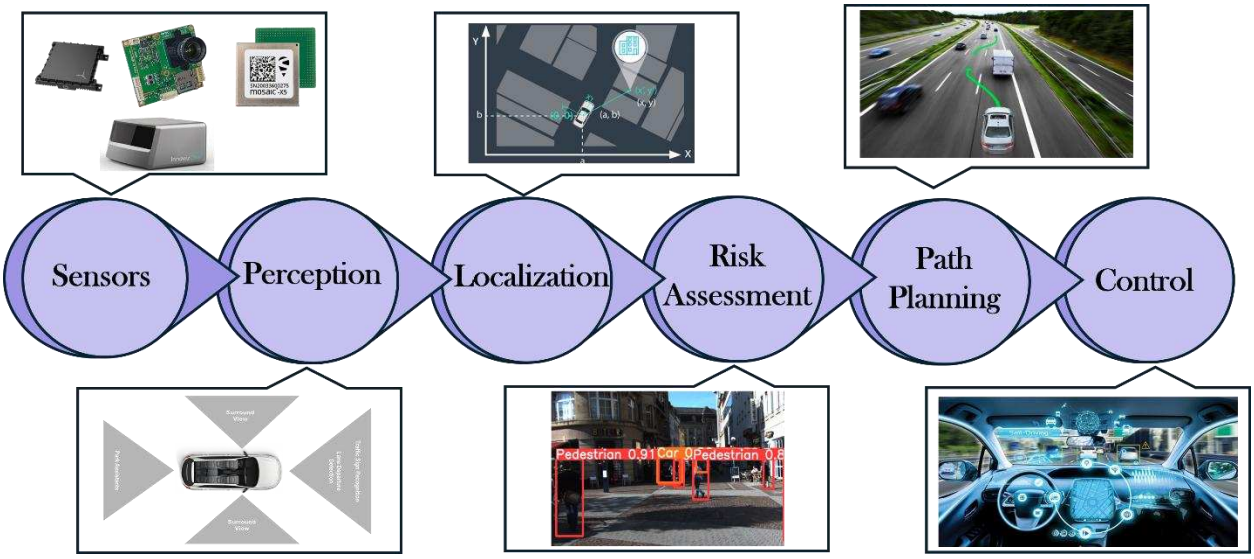


Figure 4 AVs system end to end operation flow

As seen in Figure 3, the increased adoption of multi-view camera systems and adaptive driving beams highlights the industry's shift toward sensor fusion and enhanced perception. To support this trend, future developments in object detection must focus on improving low-light performance, robustness against adversarial conditions (e.g., weather, occlusions), and computational efficiency for real-time applications.

In summary, the continued advancement of ODs is pivotal for the evolution of ADAS and AVs. As adoption rates of ADAS features rise, improvements in perception technology will drive the transition toward fully autonomous mobility, enhancing road safety and transportation efficiency.

1.2 FLOW OF OPERATION IN AUTONOMOUS VEHICLES

The AVs system comprises six distinct stages shown in Figure 4. It starts with the sensors/hardware layer, which gathers data from the environment. This is followed by the perception stage, where tasks such as object tracking, object detection, and lane detection are performed. The third stage is the localization and mapping stage, followed by the assessment stage. The fifth stage involves planning and decision-making. The final stage is the hardware control layer, which manages control actions like steering angles [9].

The AVs system begins with the hardware sensors layer, essential for interacting with the environment. These sensors collect information about the surroundings, including static and dynamic objects, which are then passed to the next layer, the perception layer. The primary goal of the perception stage is to process the data from the sensors and extract useful information for subsequent stages. The third stage, the localization and mapping layer, aims to determine the vehicle's position within a reference frame in the environment. The fourth stage, the assessment layer, focuses on overall risk estimation and predicting the intentions of surrounding human drivers to avoid accidents.

The fifth stage, path planning and decision-making, is concerned with determining the shortest, collision-free path for the vehicle in real time between the start and end points [8]. The final stage, vehicle control, involves actions such as torque, acceleration, and steering wheel angle to execute the planned path in reality. The perception architecture is crucial as it serves as the foundation for the AVs system's capability to interpret and respond to the environment. By accurately processing data from sensors, the perception stage ensures that the system can identify and track objects, detect lanes, and understand the dynamic and static aspects of the surroundings. This information is indispensable for the subsequent stages of the AVs system.

The data extracted in the perception stage is directly utilized in the assessment layer for risk estimation. By understanding the environment, the system can predict the intentions of other drivers and assess potential risks. This proactive approach to risk assessment helps in making informed decisions to avoid accidents. In the control stage, the perception data guides the hardware control actions. Accurate perception ensures that control actions, such as steering angles and acceleration, are executed precisely, resulting in safe and efficient navigation. The perception architecture, therefore, forms the backbone of both risk assessment and control in an AVs system, ensuring the vehicle's ability to navigate safely and effectively.

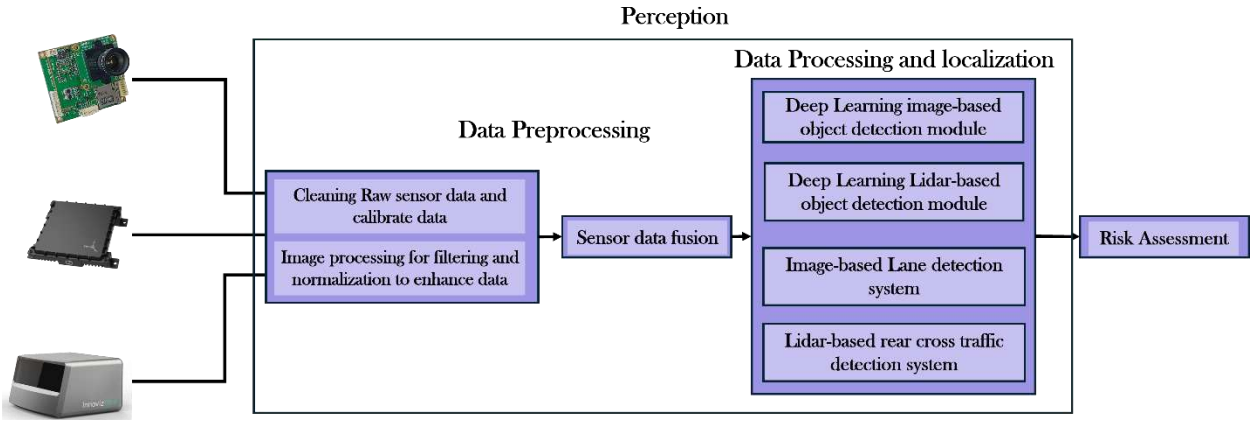


Figure 5 Perception system in AVs

1.3 PRECEPTION SYSTEM IN AUTONOMOUS VEHICLES

The perception system follows a structured flow (Figure 5) where object detection serves as a fundamental component, enabling machines to comprehend and interact with their surroundings. This process begins with data acquisition through various sensors, including cameras, LiDAR, radar, and ultrasonic sensors, which capture raw environmental data. These sensors provide diverse types of information, such as high-resolution images, depth maps, and 3D point clouds, each contributing to a more comprehensive scene understanding. Cameras offer rich visual details,

LiDAR enhances spatial awareness through precise depth estimation, radar provides robustness in adverse weather conditions, and ultrasonic sensors assist in detecting nearby objects. Integrating data from these multiple sources enhances perception accuracy, as demonstrated in multi-modal sensor fusion techniques for autonomous systems [10].

Once the raw data is collected, it undergoes preprocessing to improve quality and remove noise, ensuring that object detection algorithms operate efficiently. Preprocessing techniques such as Gaussian filtering, edge detection, and background subtraction are commonly employed to highlight essential features while suppressing irrelevant information. Feature extraction methods, including Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT), play a vital role in enhancing object representation, thereby improving detection accuracy [11]. In deep learning-based systems, convolutional neural networks (CNNs) automatically learn hierarchical features from raw data, eliminating the need for manually engineered feature extraction.

At the core of the perception system, object detection algorithms identify and classify objects present in the scene. Deep learning-based methods, such as YOLO (You Only Look Once), Faster R-CNN (Region-Based Convolutional Neural Networks), and SSD (Single Shot MultiBox Detector), are widely utilized due to their high accuracy and real-time processing capabilities. YOLO employs a single-stage detection approach, making it highly efficient for real-time applications [17]. Faster R-CNN, proposed by [12], incorporates a region proposal network (RPN) to enhance detection precision, while SSD balances speed and accuracy by using multiple feature maps for object detection at different scales [13].

After objects are detected, tracking and motion prediction algorithms are applied to analyze their movement patterns over time. Object tracking methods such as Kalman filtering and deep

learning-based tracking algorithms (e.g., SORT, DeepSORT) estimate the trajectory of moving objects, which is critical in dynamic environments like autonomous driving [14]. Motion prediction further enhances perception by forecasting future positions of detected objects, enabling proactive decision-making. For instance, in self-driving cars, predicting pedestrian movement can help the vehicle anticipate crossings and take preventive actions to avoid accidents [15].

The effectiveness of an object detection module within a perception system is highly dependent on the integration of robust detection, tracking, and prediction algorithms. Advances in deep learning, sensor fusion, and edge computing continue to push the boundaries of object detection, leading to more efficient and reliable perception systems. As autonomous systems evolve, improving object detection techniques will remain a key focus for ensuring safety, accuracy, and real-time responsiveness.

1.4 OBJECT DETECTORS IN AUTONOMOUS VEHICLES

AVs have received immense attention in recent years, in large part due to their potential to improve driving comfort and reduce injuries from vehicle crashes. It has been reported that more than 36,000 people died in 2019 due to fatal accidents on U.S. roadways [16]. AVs can eliminate human error and distracted driving that is responsible for 94% of these accidents [17]. By using sensors such as cameras, lidars, and radars to perceive their surroundings, AVs can detect objects in their vicinity and make real-time decisions to avoid collisions and ensure safe driving behavior.

AVs are generally categorized into six levels by the SAE J3016 standard [18] based on their extent of supported automation (see Table 1). While level 0 – 2 vehicles provide increasingly sophisticated support for steering and acceleration, they heavily rely on the human driver to make decisions. Level 3 vehicles are equipped with Advanced Driver Assistance Systems (ADAS) to operate the vehicle in various conditions, but human intervention may be requested to safely steer,

brake, or accelerate as needed. Level 4 vehicles are capable of full selfdriving mode in specific conditions but will not operate if these conditions are not met. Level 5 vehicles can drive without human interaction under all conditions.

Automotive manufactures have been experimenting with AVs since the 1920s. The first modern AVs was designed as part of CMU NavLab's autonomous land vehicle project in 1984 with level 1 autonomy that was able to steer the vehicle while the acceleration was controlled by a human driver [19]. This was followed by an AVs designed by Mercedes-Benz in 1987 with level 2 autonomy that was able to control steering and acceleration with limited human supervision [20]. Subsequently, most major auto manufacturers such as General Motors, Bosch, Nissan, and Audi started to work on AVs.

Table 1 SAE J3016 levels of automation

SAE Level	Name	Driving Environment monitor
0	No automation	Human Driver
1	Driver Assistance	
2	Partial Driving Automation	
3	Conditional Driving Automation	ADAS System
4	High Driving Automation	
5	Full Driving Automation	

Tesla was the first company to commercialize AVs with their Autopilot system in 2014 that offered level 2 autonomy [21]. Tesla AVs were able to travel from New York to San Francisco in 2015 by covering 99% of the distance autonomously. In 2017, Volvo launched their Drive Me feature with level 2 autonomy, with their vehicles traveling autonomously around the city of Gothenburg in Sweden under specific weather conditions [22]. Waymo has been testing its AVs since 2009 and has completed 200 million miles of AVs testing. They also launched their driverless taxi service with level 4 autonomy in 2018 in the metro Phoenix area in USA with 1000 – 2000 riders per week, among which 5 – 10% of the rides were fully autonomous without any drivers

[23]. Cruise Automation started testing a fleet of 30 vehicles in San Francisco with level 4 autonomy in 2017, launched their self-driving Robotaxi service in 2021 [9]. Even though Waymo and Cruise support level 5 autonomy, their AVs are classified as level 4 because there is still no guarantee that they can operate safely in all weather and environmental conditions.

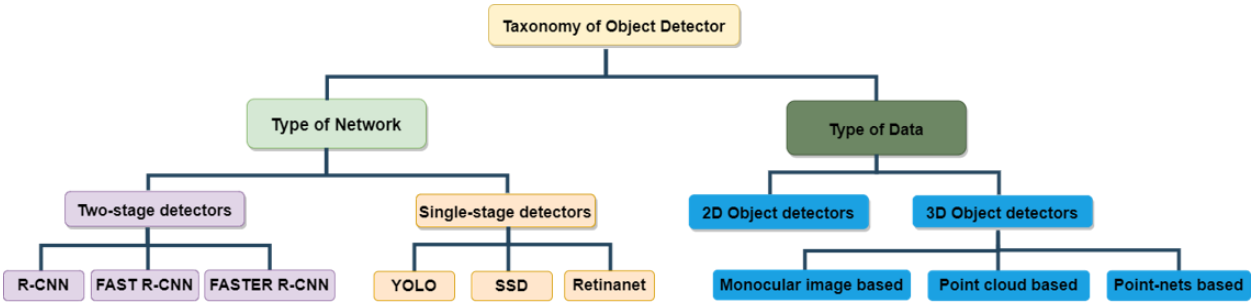


Figure 6 Taxonomy of ODs

AVs rely heavily on sensors such as cameras, lidars, and radars for autonomous navigation and decision making. For example, Tesla AVs rely on camera data with six forward facing cameras and ultrasonic sensors. In contrast, Cruise AVs use a sensor cluster that consists of a radar in the front while camera and lidar sensors are mounted on the top of the AVs to provide a 360-degree view of the vehicle surroundings [24]. One of the main tasks involved in achieving robust environmental perception in AVs is to detect objects in the AVs vicinity using software-based object detection algorithms. Object detection is a computer vision task that is critical for recognizing and localizing objects such as pedestrians, traffic lights/signs, other vehicles, and barriers in the AV vicinity. It is the foundation for high-level tasks during AVs operation, such as object tracking, event detection, motion control, and path planning.

The modern evolution of ODs began 20 years ago with the Viola Jones detector [25] used for human face detection in real-time. A few years later, Histogram of Oriented Gradient (HOG) [26] detectors became popular for pedestrian 2 detection. HOG detectors were then extended to

Deformable Part-based Models (DPMs), which were the first models to focus on multiple object detection [27]. With growing interest in deep neural networks around 2014, the Regions with Convolutional Neural Network (R-CNN) deep neural network model led to a breakthrough for multiple object detection, with a 95.84% improvement in Mean Average Precision (mAP) over the state-of-the-art. This development helped redefine the efficiency of ODs and made them attractive for entirely new application domains, such as for AVs. Since 2014, the evolution in deep neural networks and advances in GPU technology have paved the way for faster and more efficient object detection on real-time images and videos [25]. AVs today rely heavily on these improved ODs for perception, pathfinding, and other decision making. This article discusses contemporary deep learning-based ODs, their usage, optimization, and limitations for AVs. We also discuss open challenges and future directions.

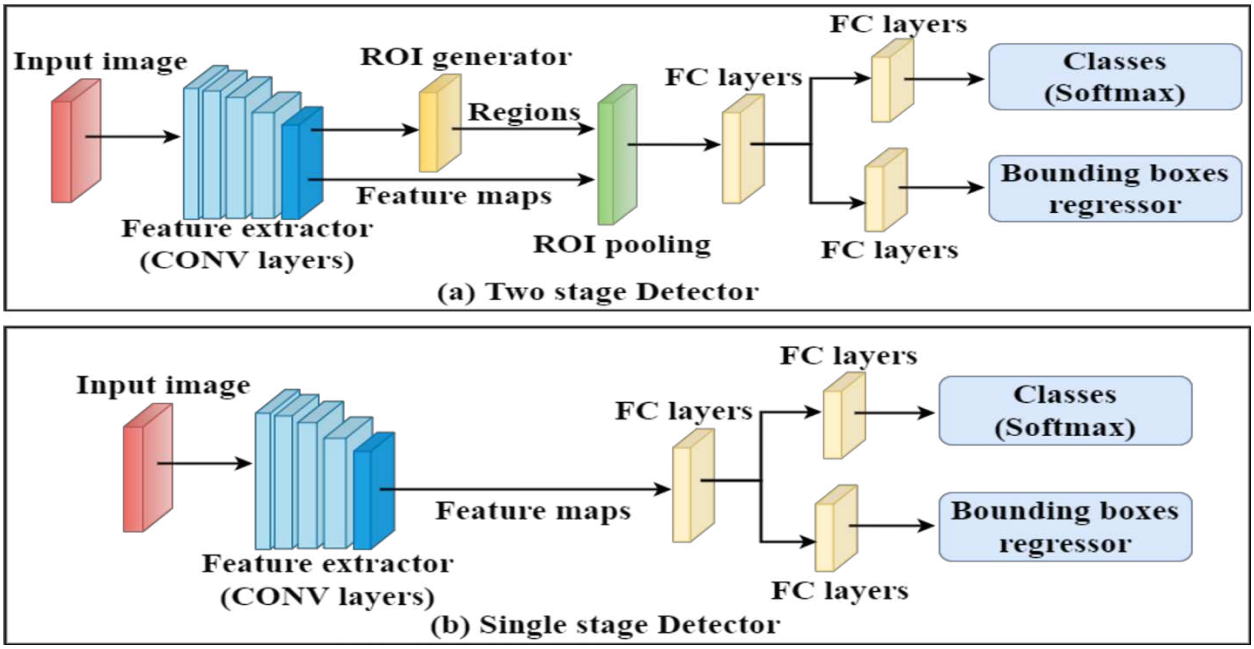


Figure 7 Two-stage vs Single stage ODs

1.5 OVERVIEW OF OBJECT DETECTORS

Object detection consists of two sub-tasks: localization, which involves determining the location of an object in an image (or video frame), and classification, which involves assigning a class (e.g., pedestrian', vehicle', traffic light') to that object. Figure 6 illustrates a taxonomy of state-of-the-art deep learning-based ODs. We discuss the taxonomy of these ODs in this section.

1.5.1 TWO-STAGE VS SINGLE STAGE OBJECT DETECTORS

Two-stage deep learning based ODs involve a two-stage process consisting of 1) region proposals and 2) object classification. In the region proposal stage, the object detector proposes several Regions of Interest (ROIs) in an input image that has a high likelihood of containing objects of interest. In the second stage, the most promising ROIs are selected (with other ROIs being discarded) and objects within them are classified [28]. Popular two-stage detectors include RCNN, Fast R-CNN, and Faster R-CNN. In contrast, single-stage ODs use a single feed-forward neural network that creates bounding boxes and classifies objects in the same stage. These detectors are faster than two-stage detectors but are also typically less accurate. Popular single-stage detectors include YOLO, SSD, EfficientNet, and RetinaNet. Figure 7 illustrates the difference between the two types of ODs. Both types of ODs are typically evaluated using the mAP and Intersection over Union (IoU) accuracy metrics. mAP is the mean of the ratio of precision to recall for individual object classes, with a higher value indicating a more accurate object detector. IoU measures the overlap between the predicted bounding box and the ground truth bounding box. Formally, IoU is the ratio of the area of overlap between the (bounding and ground truth) boxes and the area of union between the boxes. Figure 8 illustrates the IoU of an object detector prediction and the ground truth. Figure 8(a) shows a highly accurate IoU and Figure 8(b) shows a less accurate IoU.

Table 2 Different model architecture and their accuracy

Name	Year	Dataset	mAP	Inference Speed (fps)
R-CNN	2014	Pascal VOC	66%	0.02
Fast R-CNN	2015	Pascal VOC	68.80%	0.5
Faster R-CNN	2016	COCO	78.90%	7
SSD	2016	Pascal VOC	74.30%	59
YOLO	2016	Pascal VOC	63.40%	45
Complex-YOLO	2018	KITTI	64.00%	50.4
RetinaNet	2018	COCO	61.10%	90
YOLOV3	2018	COCO	44.30%	95.2
Complexer-YOLO	2019	KITTI	49.44%	100
Liu et al.	2021	KITTI	73.76%	17.8
RAANet	2021	NuScenes	62.00%	33
YOLOV5	2021	COCO	56.40%	140
DETA	2022	COCO	62.90%	12.7
M3D-RCNN	2023	KITTI	86.40%	7.6
PointRCNN++	2023	KITTI	87.00%	6.9
YOLO-NAS	2023	COCO	63.20%	115
PV-RCNN++	2024	KITTI	88.50%	10.3
RadarNeXt	2024	VoD	50.48%	28.40
Relation-DETR	2024	COCO	43.30%	7.86
YOLOv11	2024	COCO	55.00%	48.3
YOLOv12	2025	COCO	56.50%	95

R-CNN was one of the first deep learning-based ODs and used an efficient selective search algorithm for ROI proposals as part of a two-stage detection [28]. Fast RCNN solved some of the problems in the R-CNN model, such as low inference speed and accuracy. In the Fast R-CNN model, the input image is fed to a Convolutional Neural Network (CNN), generating a feature map and ROI projection. These ROIs are then mapped to the feature map for prediction using ROI pooling. Unlike R-CNN, instead of feeding the ROI as input to the CNN layers, Fast R-CNN uses the entire image Figure 6: Taxonomy of ODs 3 directly to process the feature maps to detect objects [29]. Faster R-CNN used a similar approach to Fast R-CNN, but instead of using a selective search algorithm for the ROI proposal, it employed a separate network that fed the ROI to the ROI pooling layer and the feature map, which were then reshaped and used for prediction [30].

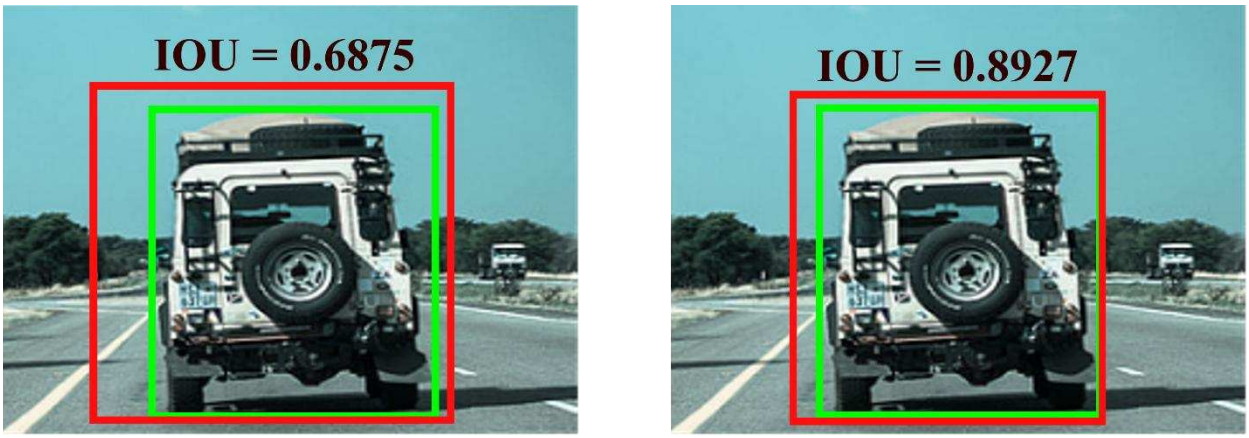


Figure 8 Example of an IOU; green box: ground truth; red box: prediction

Single-stage ODs such as YOLO (You only look once) are faster than two-stage detectors as they can predict objects on an input with a single pass. The first YOLO variant, YOLOv1, learned generalizable representations of objects to detect them faster [31]. In 2016, YOLOv2 improved upon YOLOv1 by adding batch normalization, a high-resolution classifier, and use of anchor boxes to create bounding boxes instead of using a fully connected layer like YOLOv1 [32]. In 2018, YOLOv3 was proposed with a 53 layered backbone-based network that used an independent logistic classifier and binary cross-entropy loss to predict overlapping bounding boxes and smaller objects [33]. Single-Shot Detector (SSD) models were proposed as a better option to run inference on videos and real-time applications as they share features between the classification and localization task on the whole image, unlike YOLO models that generate feature maps by creating grids within an image. While the YOLO models are faster than SSD, they trail behind SSD models in accuracy [34]. Even though YOLO and SSD models provide good inference speed, they have a class imbalance problem when detecting small objects. This issue was addressed in the RetinaNet detector that used a focal loss function during training and a separate network for classification and bounding box regression [35].

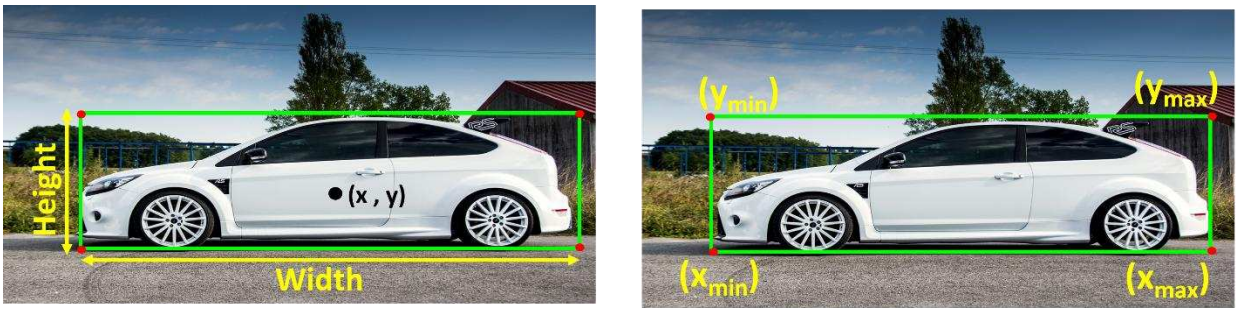


Figure 9 Commonly used bounding box encoding methods

In 2020, YOLOv4 introduced two important techniques: bag of freebies' which involves improved methods for data augmentation and regularization during training and 'bag of specials' which is a post processing module that allows for better mAP and faster inference [36]. YOLOv5, which was also introduced in 2020, proposed further data augmentation and loss calculation improvements. It also used auto-learning bounding box anchors to adapt to a given dataset [37]. Another variant called YOLOR (You Only Learn One Representation) was proposed in 2021 and used a unified network that encoded implicit and explicit knowledge to predict the output. YOLOR can perform multitask learning such as object detection, mutilabel image classification, and feature embedding using a single model [38]. The YOLOX model, also proposed in 2021, uses an anchor-free, decoupled head technique that allows the network to process classification and regression using separate networks. Unlike the YOLOv4 and YOLOv5 models, YOLOX has reduced the number of parameters and increased inference speed [39].

In 2023, YOLO-NAS (You Only Look Once - Neural Architecture Search) [40] was introduced, utilizing advanced Neural Architecture Search technology to optimize object detection models. This approach led to significant improvements in quantization support and accuracy-latency trade-offs, enhancing performance in real-time applications. YOLOv11 [41] continued the evolution of the YOLO series, focusing on integrating novel architectural innovations to improve

efficiency and precision in object detection tasks. Relation-DETR (Relation Detection Transformer) also proposed in 2024 [42], exploring explicit position relation priors for object detection. This method aimed to enhance detection accuracy by incorporating spatial relationships between objects, addressing challenges in traditional DETR architecture.

In 2025, YOLOv12 [43] was introduced, combining the fast inference speeds of CNN-based models with enhanced performance of attention mechanisms. Key innovations included the Area Attention Module (A2) and Residual Efficient Layer Aggregation Networks (R-ELAN), achieving state-of-the-art detection accuracy with lower latency. The performance of each model in terms of mAP and inference speed is summarized in Table 2.

1.5.2 2D VS 3D OBJECT DETECTORS

2D ODs typically use 2D image data for detection, but recent work has also proposed a sensor-fusion based 2D object detection approach that combines data from a camera and radar [44]. 2D ODs provide bounding boxes with four Degrees of Freedom (DOF). Figure 9 shows the most common approach for encoding bounding boxes 9(a): [x, y, height, width] and 9(b): [xmin, ymin, xmax, ymax] [45]. Unfortunately, 2D object detection can only provide the position of the object on a 2D plane but does not provide information about the depth of the object. Depth of the object is important to predict the shape, size, and position of the object to enable improved performance in various self-driving tasks such as path planning, collision avoidance, etc.

Figure 10 shows the difference between a 2D and 3D object detector output on real images. 3D ODs use data from a camera, lidar, or radar to detect objects and generator 3D bounding boxes. These detectors provide bounding boxes with (x, y, z) and (height, width, length) along with yaw information [45]. These ODs use several approaches, such as point clouds and frustum pointnets, for predicting objects in real-time. Point cloud networks can directly use 3D data, but the

complexity and cost of computing are very high, so some networks use 2D to 3D lifting while compensating for the loss of information. Pointnets are used along with RGB images, where 2D bounding boxes are obtained using RGB images. Then these boxes are used as ROIs for 3D object detection which reduces the search effort [45]. Monocular image-based methods have also been proposed that use an RGB image to predict objects on the 2D plane and then perform 2D to 3D lifting to create 3D object detection results.

Figure 10 Object detection modalities: (a) 2D vs. (b) 3D

Recent years have seen growing interest in 3D object detection with deep learning. Complex-YOLO, an extension of 4 YOLOv2, used a Euler Region Proposal Network (E-RPN), based on an RGB Birds-Eye-View (BEV) map from point cloud data to get 3D proposals. The network exploits the YOLOv2 network followed by E-RPN to get the 3D proposal [46]. Later in 2019, Complexer-YOLO achieved semantic segmentation and 3D object detection using Random Finite Set (RFS) [47]. The more recent work on 3D object detection by Wen et al. [48] in 2021 proposed a lightweight 3D object detection model that consists of three submodules: 1) point transform module, which extracts point features from the RGB image based on the raw point cloud, 2) voxelization, which divides the features into equally spaced voxel grids and then generates a many-to-one mapping between the voxel grids and the 3D point clouds, and 3) point-wise fusion module, which fuses the features using two fully connected layers. The output of the point-wise fusion

module is encoded and used as input for the model. Another 3D detector proposed in 2021 called RAANet used only lidar data to achieve 3D object detection [49]. It used the BEV lidar data as input for a region proposal network which was then used to create shared features. These shared features were used as the input for an anchor free network to detect 3D objects.

In recent years, several advancements in 3D object detection were introduced, building upon previous models to enhance performance and efficiency. In 2023, PointRCNN [50] enhanced the original PointRCNN model by introducing an adaptive deformation module and a VectorPool aggregation module. The adaptive deformation module helps the model focus on the most informative regions of point clouds, effectively handling varying object scales and densities. The VectorPool module aggregates local point features efficiently, reducing computational load while preserving spatial structure. These innovations enabled PointRCNN to achieve faster processing speeds and better accuracy. M3D-RCNN [51], also introduced in 2023, used a multi-scale, multi-modal approach by integrating LiDAR point clouds and camera images. It utilized a multi-scale feature pyramid network to fuse features from both modalities, enhancing detection across various object sizes and improving robustness and accuracy in complex environments.

In 2024, PV-RCNN++ [52] built upon the original PV-RCNN by introducing sectorized proposal-centric sampling and VectorPool aggregation. The sectorized sampling method generates more representative keypoints, while the VectorPool module further reduces resource consumption, leading to increased detection speed and accuracy. These enhancements allowed PV-RCNN++ to achieve state-of-the-art performance on large-scale datasets such as Waymo. Meanwhile, RadarNeXt, introduced in 2024 [53], integrated radar data with point cloud information for 3D object detection. By employing a two-stream network architecture, RadarNeXt processed radar and LiDAR data separately and fused them at a later stage. This design leveraged

the complementary strengths of radar and LiDAR, improving detection performance in varied weather and lighting conditions. Together, these models represent a trend of incorporating multiple data sources and refining feature aggregation techniques, resulting in advancements in 3D object detection. The performance of these models is summarized in Table 2.

1.6 DEPLOYING OBJECT DETECTORS IN AUTONOMOUS VEHICLES

Deploying deep learning-based object detector models in AVs has its own challenges, mainly due to the resource constrained nature of the onboard embedded computers used in vehicles. These computing platforms have limited memory availability and reduced processing capabilities due to stringent power caps, and high susceptibility to faults due to thermal hotspots and gradients, especially during operation in the extreme conditions found in vehicles. As the complexity of the object detector model increases, memory and computational requirements, and energy overheads also increase. In this section, we discuss techniques to improve object detector model deployment efficiency. The performance of some of the latest works on this topic is summarized in Table 3.

1.6.1 PRUNING IN OBJECT DETECTORS

Pruning a neural network model is a widely used method for reducing the model's memory footprint and computational complexity. Pruning was first used in the 1990s to reduce neural network sizes for deploying them on embedded platforms [54]. Pruning involves removing redundant weights and creating sparsity in the model by training the model with various regularization techniques (L1, L2, unstructured, and structured regularization). Sparse models are easier to compress, and the zero weights created during pruning can be skipped during inference, reducing inference time, and increasing efficiency. While most pruning approaches target deep

learning models for the simpler image classification problem, relatively fewer works have attempted to prune the more complex object detector models. Wang et al. [55] proposed using a channel pruning strategy on SSD models in which they start by creating a sparse normalization and then prune the channels with a small scaling factor followed by fine-tuning the network. Zhao et al. [56] propose a compiler aware neural pruning search on YOLOv4 which uses an automatic neural pruning search algorithm that uses a controller and evaluator. The controller is used to select the search space, pre-layer pruning schemes, and prune the model whereas the evaluator evaluates the model accuracy after every pruning step.

Table 3 Different object detector model optimization techniques and their performance

Name	Technique used	Model Compression achieved	Object Detector	Latency Improvement	Hardware used
Wang et al. [32]	Pruning	32.40 %	SSD	33.61 %	GTX 1080Ti
Zhao et al. [33]	Pruning	93.27 %	YOLOv4	80.70 %	Qualcomm Adreno 64
Fan et al. [34]	Quantization	75 %	SSDLite-MobilenetV2	85.67 %	Zynq ZC706
LCDet [24]	Quantization	77.79 %	YOLOv2	13.66 %	Snapdragon 835
Kang et al. [38]	Knowledge Distillation	37.11 %	RetinaNet	32.98 %	-
Chen et al. [39]	Knowledge Distillation	83.82 %	RCNN	7.93 %	-

1.6.2 QUANTIZATION IN OBJECT DETECTORS

Quantization is the process of approximating a continuous signal by a set of discrete symbols or integer values. The discrete set is selected as per the type of quantization such as integer, floating-point, and fixed-point quantization. Quantizing deep learning-based object detector models involves converting the baseline 32-bit parameters (weights, activations, biases) to fewer (e.g., 16 or 8) bits, to achieve lower memory footprint, without significantly reducing model

accuracy. Fan et al. [57] proposed an 8-bit integer quantization of all the bias, batch normalization, and activation parameters on SSDLiteMobileNetV2. LCDet [58] proposed a fully quantized 8-bit model in which parameters of each layer of a YOLOv2 object detector were quantized to 8-bit fixed point values. To achieve this, they first stored the minimum and maximum value at each layer and then used relative value to linearly distribute the closest integer value to all the reduced bitwidth weights.

1.6.3 KNOWLEDGE DISTILLATION IN OBJECT DETECTORS

Knowledge Distillation involves transferring learned knowledge from a larger model to a smaller, more compact model. A teacher model is first trained for object detection, followed by a smaller student model being trained to emulate the prediction behavior of the teacher model. The goal is to make the student model learn important features to arrive at the predictions that are very close to that of the original model. The resulting student model reduces the computational power and memory footprint compared to the original teacher model. Kang et al. [59] proposed an instance-conditional knowledge decoding module to retrieve knowledge from the teacher network (RetinaNet with a ResNet-101 classifier model as backbone) via query-based attention. They also used a subtask that optimized the decoding module and feature maps to update the student network (RetinaNet with a simpler ResNet-50 model as backbone). Chen et al. [60] proposed a three-step knowledge distillation process on R-CNN with a Resnet-50 model as backbone. The first step used a feature pyramid distillation process to extract the output features that can mimic the teacher network features. They then used these features to remove the output proposal to perform Regional Distillation (RD), enabling the student (RCNN with a much simpler ResNet-18 model as backbone) to focus on the positive regions. Lastly, Logit Distillation (LD) on the output was used to mimic the final output of the teacher network.

1.7 OPEN DATASET FOR OBJECT DETECTORS

Object detection model performance can significantly vary due to changing lighting, weather, and other environmental conditions. To address this challenge, incorporating data from different weather conditions during training can help the model adapt to various environmental factors. By adding new data that accommodates these varying weather conditions when training and testing, we can enhance the robustness and reliability of ODs. Open datasets that focus on different lighting and weather conditions, such as the Waymo Open Dataset, serve as excellent examples of how this can be achieved. However, the availability of more open datasets covering a range of environmental conditions is essential to train reliable ODs for Avs and ensure robust performance across all scenarios.

One such example is the Waymo Open Dataset [61], which offers a wide variety of data, including different lighting and weather conditions, to tackle environmental challenges in autonomous driving. This dataset includes lidar, camera images, and radar data, with annotations for object detection, tracking, and segmentation tasks. It is ideal for training models that need to perform well in diverse weather conditions, such as rain, fog, and varying lighting scenarios. However, datasets like this are still limited, and the availability of additional datasets will be crucial for training models capable of performing well under all environmental conditions.

Another important dataset is the KITTI Vision Benchmark Suite [62], widely used for evaluating object detection, stereo, and optical flow tasks in autonomous driving. It provides stereo camera images, lidar data, and GPS/IMU data, primarily focused on urban environments with good lighting conditions. Although KITTI does not offer extensive data on extreme weather conditions, it remains a valuable resource for training ODs in clear weather with moderate lighting variation. Similarly, the Cityscapes dataset focuses on urban scene understanding, offering high-quality

annotations for objects like vehicles and pedestrians in daytime conditions. It lacks significant weather variation but is still valuable for urban object detection tasks.

The nuScenes dataset [63] is another significant resource, offering a 360-degree view through lidar, radar, and camera data collected from real-world driving environments. It includes data from varying weather conditions, including rain, fog, and different times of the day. This dataset is particularly useful for training models that need to handle diverse environmental challenges. Similarly, the Argoverse dataset [64] provides data for 3D tracking and 2D segmentation in urban environments, with data from various weather conditions such as clear, rainy, and foggy weather, making it suitable for training models to handle different environmental factors.

The ApolloScape dataset [65] offers large-scale data for autonomous driving research, with annotations for segmentation, object detection, and tracking tasks. While it doesn't have as wide a variety of weather data as other datasets, it does include some data for varying lighting and weather conditions, such as nighttime and rainy weather. For pedestrian detection tasks, the Daimler MonoPedestrian dataset [66] provides monocular data with high-resolution images and detailed annotations for pedestrians in urban traffic situations, though it lacks a broad range of weather variations. The BDD100K dataset [67] stands out due to its size, including over 100,000 images with diverse weather conditions such as sunny, rainy, foggy, and nighttime driving. This dataset is excellent for training models on a variety of environmental challenges.

In conclusion, a diverse range of datasets with different environmental conditions is crucial for developing robust object detection models for Avs. Datasets like Waymo, nuScenes, and BDD100K are particularly valuable, as they provide data that reflects varying weather conditions and lighting, which is essential for training models that need to perform reliably in all real-world

scenarios. However, more such datasets are needed to ensure that object detection systems are adaptable to all environmental factors.

1.8 OPEN CHALLENGES AND OPPORTUNITIES

While there has been significant work on effective object detection for AVs, there are significant outstanding challenges that remain to be solved. Here we discuss some of the key challenges and opportunities for future research in the field.

1.8.1 REAL-TIME PROCESSING

ODs deployed in AVs use video inputs from AV cameras, but the ODs are typically trained to detect objects on image datasets. This discrepancy between training data and real-world input can introduce challenges in the detection process. Detecting an object on every frame in a video can significantly increase the latency of the detection task, as the computational load is high. This delay can impact on the overall performance and responsiveness of the AV, which is critical for safe and efficient operation.

However, correlations between consecutive frames can help address this issue. By identifying the frames that can be used for detecting new objects and discarding others, the latency of the model can be reduced. This approach leverages the temporal continuity in video streams to optimize the detection process. Creating models that can effectively correlate spatial and temporal relationships between consecutive frames remains an open problem in the field of computer vision and autonomous driving. Developing such models requires advanced techniques that can accurately capture the dynamics of moving objects and scenes in real-time.

Recent work on real-time object detection [68], [69], has begun to address this problem, but much more work is needed. These efforts have shown promising results in reducing latency and

improving detection accuracy by incorporating temporal information. However, the complexity of real-world scenarios and the need for robust performance under varying conditions necessitate further research. Future studies should focus on enhancing the ability of ODs to process video inputs efficiently while maintaining high accuracy. Additionally, exploring new algorithms and frameworks that can seamlessly integrate spatial and temporal data will be crucial for advancing real-time object detection in AVs.

The development of efficient and reliable object detection systems is essential for the widespread adoption of AVs. By addressing the challenges associated with video-based object detection, researchers can contribute to the progress of AV technology and its potential to transform transportation.

1.8.2 SENSOR FUSION

Sensor fusion is one of the most widely used methods for increasing the accuracy of 2D and 3D object detection. By combining data from multiple sensors, such as lidar and RGB images, object detection systems can benefit from the complementary strengths of each sensor type, leading to more robust and accurate detection capabilities. Many efforts have focused on fusing lidar and RGB images to perform object detection for autonomous driving, as this combination provides rich spatial and visual information that enhances the perception of the environment.

However, there are very few works that consider fusion data from ultrasonic sensors, radar, or V2X (Vehicle-to-Everything) communication. The integration of these additional sensor types is vital for increasing the reliability of the perception system in AVs . Ultrasonic sensors, radar, and V2X communication can provide valuable information that complements lidar and RGB images, further improving the overall performance of the perception system. For example, radar can provide accurate distance measurements and work effectively in adverse weather conditions,

while V2X communication can enable vehicles to share information with each other and with infrastructure.

Fusing data from a more diverse set of sensors can also increase the stability of the perception system and ensure that it does not fail when one of the sensors is compromised due to environmental conditions. This redundancy is crucial for the safe and reliable operation of AVs, as it allows the system to maintain accurate perception even in challenging scenarios. By leveraging data from multiple sensors, the perception system can compensate for the weaknesses of individual sensors and provide a more comprehensive understanding of the environment.

Recent efforts [70], [71] are beginning to design ODs that work with data from various sensors, which is a step in the right direction for reliable perception in AVs. These efforts are paving the way for more advanced and resilient object detection systems that can handle the complexities of real-world environments. As research in this area continues to progress, we can expect to see further advancements in the fusion of data from diverse sensors, ultimately leading to safer and more reliable AVs.

1.8.3 RESOURCE CONSTRAINTS

Most ODs have high computational and power overheads when deployed on real hardware platforms. To address this challenge, prior efforts have adopted techniques such as pruning, quantization, and knowledge distillation (see Section 1.6) to reduce model footprint and decrease computational needs. These methods help streamline the models, making them more efficient and less resource intensive.

New approaches for hardware-friendly pruning and quantization, as demonstrated in recent efforts [72], [73], are proving to be very useful. These techniques focus on simplifying the model without significantly sacrificing performance. Additionally, methods to reduce matrix

multiplication operations, such as those proposed in [74], [75], [76], can further speed up object detector execution time by optimizing the computational processes involved.

Hardware and software co-design, which combines pruning, quantization, and knowledge distillation with hardware optimization techniques such as parallel factors adjustment and resource allocation, represents a comprehensive approach to improving object detector efficiency. This integrated strategy ensures that both the software algorithms and the hardware resources are optimized to work together seamlessly.

Results from recent work [77], [78], [79] have been promising, showing significant improvements in model efficiency and performance. However, much more research is needed to further advance these techniques and develop new solutions that can overcome the remaining challenges.

1.9 THESIS OVERVIEW

In summary, optimizing object detection models for AVs is essential to improve performance and reliability. Current models face challenges in computational complexity and real-time inference, affecting robustness in varied driving environments. To tackle these issues, pruning, quantization, and knowledge distillation reduce model footprint and computational needs. New approaches for hardware-friendly pruning and quantization [72], [73] and minimizing matrix multiplication operations [74], [75], [76] show promise in speeding up object detection. Integrating data from different sensor types and using temporal information can enhance robustness and accuracy. However, transformer-based models, despite their high accuracy, pose deployment challenges due to large memory requirements. Recent efforts [77], [78], [79] have made progress, but more research is needed. The main contribution of this thesis is the design of two novel optimization techniques, *R-TOSS* and *UPAQ*. The rest of the thesis is organized as follows:

In chapter 2, we present *R-TOSS*, a novel semi-structured pruning framework designed to reduce the computational and memory overheads of ODs used in AVs. *R-TOSS* addresses the shortcomings of state-of-the-art model pruning techniques by employing kernel pruning without connectivity pruning to preserve kernel information for inference.

In chapter 3, we present *UPAQ*, a novel compression framework that employed a kernel transformation technique that uses a mixed precision quantization along with pruning for 3D pointcloud LiDAR data. We also introduce a model of on-device efficiency of the compressed model to select the best fit quantized kernels for the model.

Chapter 4 concludes this thesis. We summarize our comprehensive body of research in this chapter and also make recommendations for future work

2 R-TOSS: A FRAMEWORK FOR REAL-TIME OBJECT DETECTION USING SEMI-STRUCTURED PRUNING

ODs used in AVs can have high memory and computational overheads. In this chapter, we introduce a novel semi-structured pruning framework called *R-TOSS* that overcomes the shortcomings of state-of-the-art model pruning techniques. Experimental results on the JetsonTX2 show that *R-TOSS* has a compression rate of 4.4× on the YOLOv5 object detector with a 2.15× speedup in inference time and 57.01% decrease in energy usage. *R-TOSS* also enables 2.89 × compression on RetinaNet with a 1.86 × speedup in inference time and 56.31% decrease in energy usage. We also demonstrate significant improvements compared to various state-of-the-art pruning techniques.

2.1 INTRODUCTION AND CONTRIBUTION

In recent years, AVs have received immense attention due to their potential to improve driving comfort and reduce injuries from vehicle crashes. A report from the National Highway Traffic Safety Administration (NHTSA) indicated that in 2021, more than 31,720 people were involved in fatal accidents on U.S. roadways [80]. These accidents were found to predominantly be caused by distracted drivers who contributed to ~94% of them. AVs can help mitigate human errors and avoid such accidents with the help of their superior perception systems. A perception system helps AVs understand their surroundings with the help of an array of sensors that can include Lidars, Radars, and Cameras. Object detection is an important component of such perception systems [17]. AVs must process a huge amount of data in real-time to provide precise corrections to the vehicle controller to maintain its course, speed, and direction. To assist with vehicle path planning and control, AVs rely on ODs to provide information about the obstacles in

their surroundings. These ODs must satisfy two important conditions: 1) maintaining high accuracy, and 2) providing inference in real-time (~tens of milliseconds). In recent years researchers have been able to design machine learning models for object detection with high accuracy, but these models are generally very compute-intensive and often combined with sensor fusion task which helps in providing the input to these models by combining data from various sensors [81][106]. Apart from these ODs Avs also must process immense data as a part of the Advance Driver Assistance System (ADAS) for operation safety and security such as in-vehicle communication and vehicle-to-x (V2X) protocols which can increase the computational cost and power usage [82][97][104]. This is a challenge because onboard computers in AVs are resource-constrained, with strict limits on power dissipation and computational capabilities. Object detection is a compute and memory-intensive task involving both classification and regression. Typically, all machine learning based ODs can be classified into two types: 1) Two-stage detectors and 2) Single-stage detectors. Two-stage detectors consist of a two-stage detection process that involves a region proposal stage and subsequent object classification stage. The regional proposal stage often consists of a Regional Proposal Network (RPN) which proposes several Region of Interests (ROIs) in an input image (e.g., from a camera sensor in an AVs). These ROIs are used to classify objects in them. The objects are then surrounded by bounding boxes to localize them. Examples of two-stage detectors include R-CNN [28], Fast R-CNN [29], and Faster R-CNN [30]. In contrast to two-stage detectors, single-stage detectors use a single feed-forward network which involves both classification and regression to create bounding boxes to localize objects. Singlestage detectors are lightweight and faster than two-stage detectors. Some examples of single-stage detectors are YOLOv5 [37] (You Only Look Once), RetinaNet [35], YOLOR [38], and YOLOX [39]. Unfortunately, even single-stage detectors are compute and memory intensive, so

deploying and executing them on embedded and IoT boards in AVs remains a bottleneck [83]. To address this bottleneck, many techniques have been proposed in recent years, such as pruning, quantization, and knowledge distillation, to compress and optimize object detector execution, with an emphasis on improving inference time while preserving model accuracy. Pruning techniques in particular have been shown to be very effective in increasing the sparsity of object detector models, by carefully removing redundant weights that do not impact overall accuracy. Such sparse models require fewer computations, and can be compressed to reduce latency, memory, and energy costs. In this chapter, we introduce the *R-TOSS* object detector pruning framework to achieve efficient pruning of ODs used in AVs. Unlike traditional pruning algorithms that can generally be classified as structured pruning [90]-[94] or unstructured pruning [84]-[89], we utilize a more niche approach that involves semi-structured pruning. Our approach involves applying specific kernel patterns to prune convolutional kernels and associated connectivity. The novel contributions of our proposed pruning framework for ODs are as follows:

- An approach for reducing computational cost of iterative pruning by using depth first search to generate parent-child kernel computational graphs, to be pruned together,
- A pruning technique to prune 1×1 kernel weights to increase achievable model sparsity,
- An implementation of kernel pruning without connectivity pruning, to preserve kernel information for inference, that can help retain model accuracy,
- A detailed comparison against multiple state-of-the-art pruning approaches to showcase the effectiveness of our novel framework, in terms of mAP, latency, energy usage, and achieved sparsity.

The rest of the chapter is organized as follows: Section 2.2 provides a detailed background in terms of state-of-the-art object detector models and pruning techniques; Section 2.3 describes

the motivation for this work; Section 2.4 introduces our framework and provides a deep dive into the algorithms proposed; Section 2.5 showcases our experimental results, and Section 2.6 presents our conclusions.

2.2 BACKGROUND AND RELATED WORK

2.2.1 OBJECT DETECTORS

ODs are used in AVs for various tasks such as traffic sign and traffic light recognition, lane recognition, vehicle tracking, etc. These ODs can be classified into two categories, twostage and single-stage detectors. To evaluate an object detector, irrespective of the category, mean average precision (mAP) and intersection over union (IoU) metrics are used. mAP is the mean of the ratio of precision to recall for individual object classes, with a higher value indicating a more accurate object detector. IoU measures the overlap between the predicted bounding box and the ground truth bounding box.

2.2.1.1 TWO-STAGE OBJECT DETECTORS

Two-stage detectors use a two-stage process consisting of a region proposal and object classification. RCNN [28] was one of the first deep learning-based ODs to be proposed. The algorithm's novelty came with an efficient selective search algorithm for ROI proposals, which dramatically decreased the overall number of regions needed to be processed. The regions were fed into a convolutional neural network (CNN) for feature extraction. The CNN output was sent to a support vector machine (SVM) for classifying the object in the region. Even though the reduction in ROI proposals was revolutionary in terms of minimizing inference, the R-CNN algorithm cannot infer in real time, as it takes ~40s to process a single input image. To address the latency challenge, the same authors proposed Fast R-CNN [29]. In Fast R-CNN, a CNN is used to

generate convolution feature maps of the input images rather than for feature extraction. The feature maps are used for ROI identification and the ROIs are warped to squares using a pooling layer, which is further transformed into a vector to be fed into a fully connected (FC) layer. The feature vector from the FC layer is used for object class prediction in the ROI, using a softmax layer, and a bounding box regressor is used for coordinating prediction. Fast R-CNN exhibited inference speeds around ~2s, significantly faster than R-CNN, but the latency is still high, making it unusable in a real-time scenario. Faster R-CNN [30] tackled the high latency caused by the region proposal mechanism in both the prior R-CNN works, by directly feeding the image to the CNN and letting the CNN learn to perform ROI prediction. This remarkably reduced the latency to ~0.2s. Despite their desirable accuracies, two-stage detectors are bulky and have best case latencies in the hundreds of milliseconds on highend GPUs. These latencies and resource overheads make them impractical for embedded real-time use cases, such as in AVs.

2.2.1.2 SINGLE-STAGE DETECTORS

Single-stage detectors are much faster than two-stage detectors because they use a single feed-forward network without any intermediate stage for ROI proposals. The YOLO algorithm was revolutionary when it came out in 2016 as it reframed object detection as a single-stage regression problem, from image feature extraction, to bounding box generation, and object classification. The follow-up variants of YOLO made it faster and more accurate while preserving the single shot detection philosophy. YOLOv4 introduced two important techniques: bag of freebies' (BoF) which involves improved methods for data augmentation and regularization during training and 'bag of specials' (BoS) which is a post processing module that leads to better mAP and faster inference [36]. YOLOv5 [37] proposed additional data augmentation and loss calculation improvements. It also used auto-learning bounding box anchors to adapt to a given

dataset. Even though YOLO models provide good inference speed, they have a class imbalance problem when detecting small objects. This issue was addressed in the RetinaNet [35] detector that used a focal loss function during training and a separate network for classification and bounding box regression. Table 4 shows a comparison of various two-stage and single stage ODs on the COCO dataset [95].

Table 4 Metrics comparison of two-stage vs single-stage detectors

Name	Type	mAP	Inference rate (fps)
R-CNN [13]	two-stage	42%	0.02
Fast R-CNN [14]	two-stage	19.7%	0.5
Faster R-CNN [15]	two-stage	78.9%	7
RetinaNet [20]	single-stage	61.1%	90
YOLOv4 [21]	single-stage	65.7%	62

While single-stage detectors are faster than two-stage detectors, they still incur significant inference times when deployed on an embedded board. To further reduce latency, model compression techniques, such as pruning, quantization, and knowledge distillation, are essential to consider. Quantization requires specialized hardware support for efficient deployment, which may not be available on embedded boards. Knowledge distillation requires the student model to be robust in order to absorb and retain the information from the teacher model, which requires both time and complex computation. Compared to its counterparts, pruning is neither computationally complex nor hardware bound, so we focus on pruning for accelerating object detector inference in this work.

2.2.2 DNN MODEL PRUNING

Pruning an object detection model aims to reduce model footprint and computational complexity by removing weight parameters from the model using some criteria. Consider a deep

learning model with L_n number of layers. The most compute-intensive operation of a deep learning model is the Convolution (Conv) layer. If each Conv layer has k_n number of kernels with W_n number of non-zero weights, during inference, the computational cost of the model is a function of $(W_n \times k_n \times L_n)$. This computational cost increases dramatically as the parameters involved increases, as is the trend in modern deep learning models. By performing parameter pruning, we can induce sparsity in the model which will decrease the parameters in W_n and through kernel pruning we can also decrease k_n. This decreases the overall computational cost. Emerging computing platforms provide software compression techniques [98] which can compress the input and weight matrices in response to the presence of zero valued (pruned) parameters, thus skipping them entirely during model execution. The skipping operation may optionally also be performed by the hardware with specifically designed hardware [99]. Pruning approaches from prior work can be classified into three major categories: unstructured pruning, structured pruning, and semi-structured or pattern-based pruning.

2.2.2.1 UNSTRUCTURED PRUNING

In unstructured pruning, redundant weights (Figure. 11(a)) are pruned opportunistically, while keeping the loss to minimum which helps in retaining the accuracy of the model. Several unstructured pruning schemes have been proposed, such as: 1) weight magnitude pruning, that focuses on replacing a set of weights below a predefined threshold to zero [84], [85]; 2) gradient magnitude pruning, that prunes a set of weights whose gradients are below a predefined threshold [86], [87]; 3) synaptic flow pruning, which is an iterative pruning technique that uses a global scoring scheme and prunes a set of weights until the global score drops below a threshold [88]; 4) second order derivative pruning, that calculates the second order derivative of weights by replacing a set of weights by zero and keeping the loss of the network close to the original loss [89]. These

approaches negatively impact thread level parallelism due to the load imbalance caused by different levels of sparsity across weight matrices. Irregular sparsity also affects memory performance due to changes it creates in data locality, leading to reduced benefits from caching across various platforms (GPUs, CPUs, TPUs).

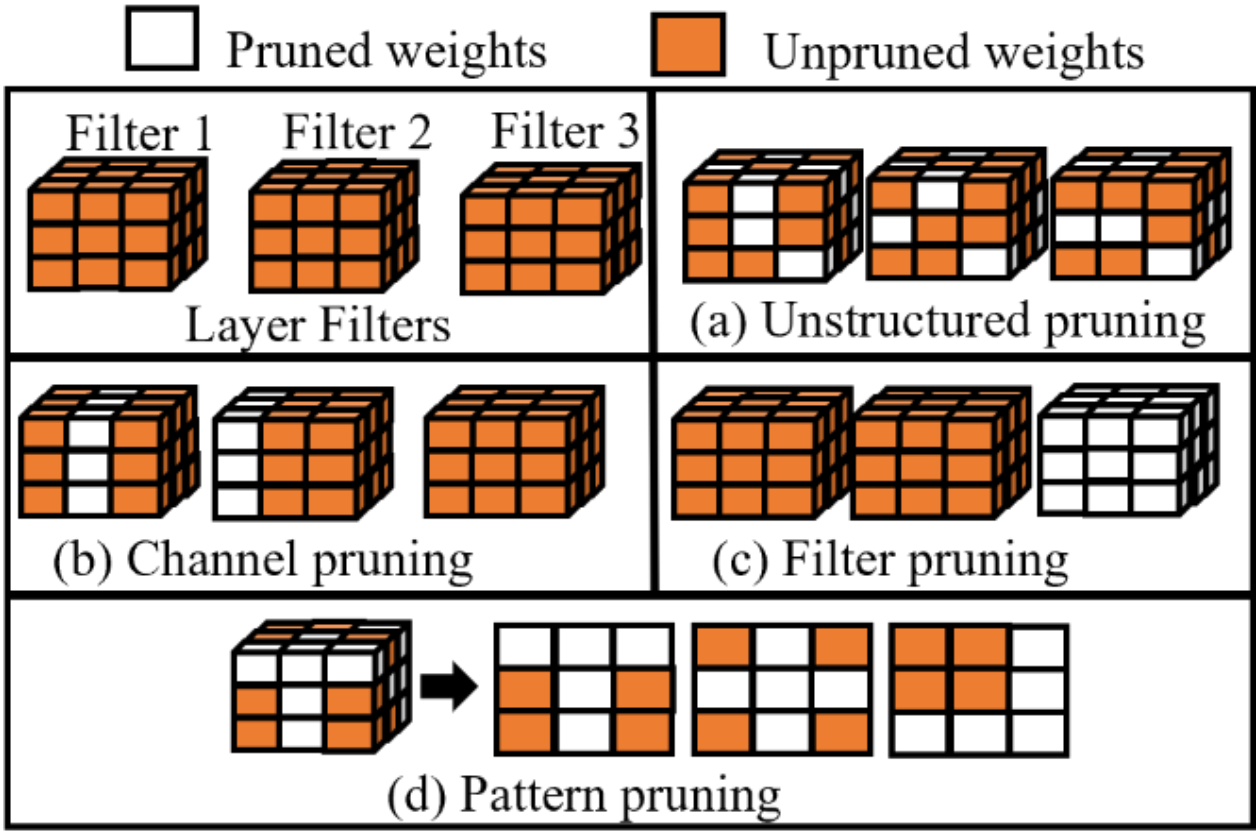


Figure 11 Illustration of different pruning methods

2.2.2.2 STRUCTURED PRUNING

In structured pruning, an entire filter (Figure. 11(c)) [90]-[92] or consecutive channels (Figure. 11(b)) [93], [94] are pruned, to increase sparsity of the model. Filter/channel pruning provides a more uniform weight matrix and reduces the size of the model. The reduced matrix sizes help in reducing the number of multiply and accumulate (MAC) operations compared to that of unstructured pruning. However, structured pruning also decreases the accuracy of the model

since weights that can be contributing to the overall accuracy of the model will also be pruned along with the redundant weights. Structured pruning can also be used with acceleration platforms like TensorRT [96]. Unlike unstructured pruning, due to the uniform nature of the weight matrix, structured pruning can better utilize the hardware acceleration provided by various platforms in terms of memory and bandwidth [92], [94].

2.2.2.3 SEMI-STRUCTURED PRUNING

Semi-structured pruning, also called pattern pruning, is a combination of structured pruning and unstructured pruning schemes (Figure. 11(d)). This type of pruning utilizes kernel patterns that can be used as a mask on a kernel. A mask prevents the weights it covers from being pruned, inducing partial sparsity in a kernel. By evaluating the effectiveness of the pruned kernel, by utilizing L2 norm for example, the most effective pattern masks can be identified and deployed during inference. Since the kernel patterns can only prune a fixed number of weights inside a kernel, they will induce lesser sparsity than that of its counterparts [100], [101]. To overcome this issue, pattern pruning is applied together with connectivity pruning which prunes some of the kernels entirely. However, most modern ODs have a large number of 1×1 kernels which contain redundant weights that are not pruned during this process. This is because, pattern pruning techniques typically focus on kernels with sizes 3×3 and above, that have more candidate weights for pruning. Connectivity pruning also reduces the accuracy of the model since several important weights in a particular kernel are also removed during this process. However, kernel pattern pruning due to its semi-structured nature can still leverage hardware parallelism to reduce inference times of the model [101].

2.3 MOTIVATION

ODs designed for use in AVs require high accuracy, but consequently, these models also have overheads such as a large memory footprint and higher inference time [108]. To overcome these issues, we need to come up with a model that can be lightweight and can achieve high accuracy. Single-stage detectors such as YOLOv5, RetinaNet, Detection Transformer (DETR), and YOLOR are good starting points to achieve real-time object detection goals, but these models still have a high memory footprint which can decrease model performance. Table 5 summarizes the inference time as the size of the object detector model increases, on a Jetson TX2 platform.

Table 5 Comparison of model sizes vs. execution time

Models	Number of parameters (Millions)	Execution time (sec)
YOLOv5 [64]	7.02	0.7415
YOLOX [24]	8.97	1.23
RetinaNet [20]	36.49	6.8
YOLOv7 [86]	36.90	6.5
YOLOR [23]	37.26	6.89
DETR [84]	41.52	7.6

In order to reduce latency of operation, while retaining model accuracy a pruning technique can be employed. Among pruning techniques, pattern-based semi-structured pruning can offer better sparsity over unstructured pruning, while ensuring better accuracy than structured pruning techniques. Semi-structured pruning also allows for more regular weight matrix shapes, allowing the hardware to better accelerate the model inference. Simultaneously, it does not prune entire kernel weights, unlike structured pruning, thus retaining more information and, hence ensuring better accuracies. Therefore, pattern-based pruning techniques can generate models with high sparsity and high accuracy, ideally.

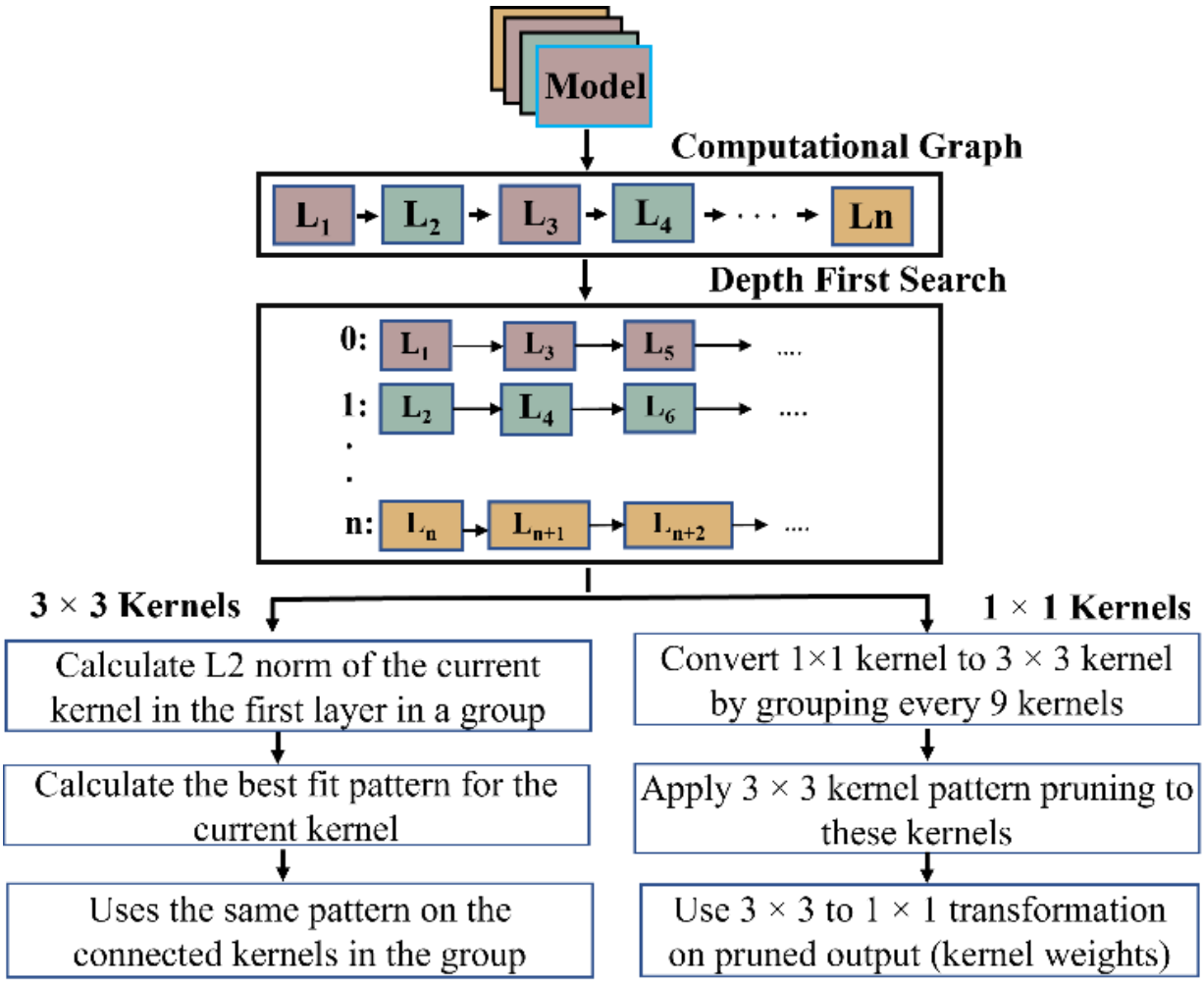


Figure 12 An overview of the proposed R-TOSS pruning framework

However, a caveat of pattern-based pruning, which limits the achievable sparsity and hence the inference acceleration benefits, is that current techniques primarily focus on 3×3 kernels. Most state-of-the-art models such as YOLOv5, RetinaNet and DETR consist of 68.42%, 56.14% and 63.46% of small 1×1 kernels, respectively. So, to increase the sparsity of such models, pattern-based pruning techniques sometimes employ connectivity pruning on these 3×3 kernels [100]. But the 'last kernel per layer' criteria used in connectivity pruning contributes to loss of important information which can affect the accuracy of the model. So, we elect to avoid connectivity pruning in our pruning framework. Moreover, this technique still does not address the 1×1 kernels, which

constitute a significant portion of the kernels, as mentioned above.

ALGORITHM 1: LAYER GROUPING USING DFS

Inputs: *Pretrained model (M)*
compute the computational graph (G) of the pretrained model
group_list ← Ø
for *l* in *M*:
  apply DFS with G and find parent [l_p] of M[l_c]
  if *group_list[l_p[0]]* **then**
    group_list[l_p] ← *M[l_c]*
  else
    group_list[l_p] ← *0*
    group_list[l_p] ← *M[l_c]*
  end
end
Output: *a list of parent-child layer groups (group_list)*

To address these shortcomings, we propose a three-step pruning approach to prune 1×1 kernels: 1) group 1×1 kernels to form 3×3 temporary weight matrices; 2) apply kernel pattern pruning on these weight matrices; 3) decompose the temporary weight matrices to 1×1 kernels and reassign to their original layers. Our approach thus increases the sparsity of the model while preserving important information which contributes to the accuracy of the model.

2.4 *R-TOSS* PRUNING FRAMEWORK

In this section, we describe our novel *R-TOSS* pruning framework and detail how we have implemented the previously mentioned improvements to the kernel pruning technique on the YOLOv5 and RetinaNet ODs. A straightforward approach to pruning, while retaining much of the original performance of the model, is to adopt an iterative pruning approach. But this is a naïve approach as an iterative approach can quickly become unwieldy in terms of computational cost and time requirement as the model sizes increase. As mentioned in Section 2.3.3, the model sizes

of modern ODs are increasing, but for many application spaces which employ them, such as AVs, their accuracy cannot be compromised.

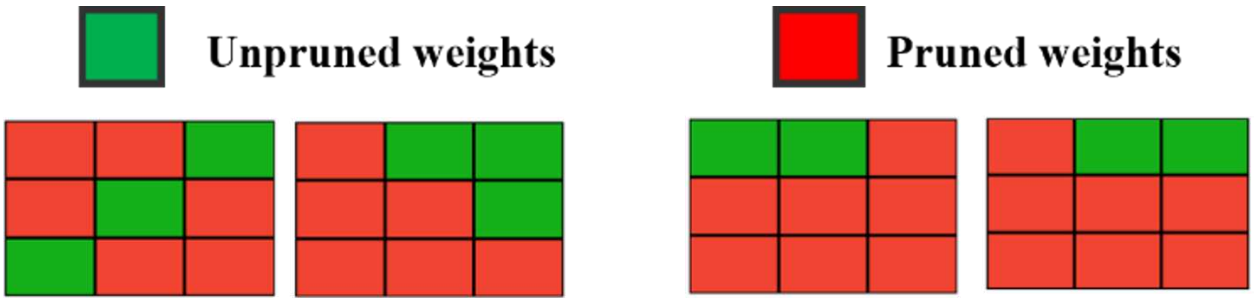
ALGORITHM 2: 3×3 KERNEL PRUNING

Inputs: *3×3 Kernel Weights (K_W)*

```
shape ← K_W.shape
kernel_patterns_dict ← patterns used to prune the kernel.
for i in range (shape[0]):
    for j in range (shape[1]):
        temp_kernel = K_W[i, j, :, :].copy()
        L2_dict ← Ø
        for key, pattern in kernel_patterns_dict.items():
            for index in pattern:
                L2_norm = L2.norm(temp_kernel)
                L2_dict[key] = L2_norm
            bestfit ← best fit pattern for the kernel in terms of L2norm
            for index in patterns_dict1[bestfit]:
                K_W[i, j, index[0], index[1]] = 1
            end
        end
    end
end
```

Outputs: *Pruned 3×3 kernels.*

Our *R-TOSS* framework (Figure 12) adopts an iterative pruning scheme with several optimizations for reducing computational cost and time overheads. We start by using a depth first search (DFS) algorithm which is used to find the parent-child layer couplings within the model. The parent-child graphs thus obtained are used to reduce the computation requirements for pruning. The reduction in computation costs happens as the pruning at the parent layer is reflected in its child layers within the graph. We follow up DFS with identifying the 3×3 and 1×1 kernels within the sub-graphs and applying kernel size specific pruning to them. These algorithms are discussed in detail, in the following subsections.



(a) Examples of 3-entry patterns (b) Examples of 2-entry patterns

Figure 13 Illustration of kernel patterns

2.4.1 DFS ALGORITHM

Algorithm 1 shows the pseudocode for the DFS algorithm. Using the pretrained model as input, we compute the computational graph (G) using the gradients obtained from backpropagation. An empty list (group_list) is initialized (line 2) to store the parent-child layer groups. We then traverse the model layers (l) and apply DFS search on the computational graph G to identify the parent of that layer. If a layer does not have any parent layer, then we assign that layer as its own parent layer (lp) (lines 7–9) and this becomes a group. If a layer is identified as the child layer (lc) to any layer in the group_list (line 5) then this layer now becomes the parent layer (lp) of the child layer (lc) and added to that group (lines 5–6). Each parent layer (lp) can have multiple child layers (lc) but each child layer can only have one parent layer (lp). This process continues until all the layers are assigned to a group. Since layers in each group have coupled channels in them, they also share their kernel properties, hence they can share the same kernel patterns.

2.4.2 SELECTING KERNEL PATTERNS

We generate pattern masks in all possible combinations via standard combinatorics, using the following equation:

$$n(k) = \binom{n}{k} = \frac{n!}{k!\,(n-k)!} \tag{1}$$

where, *n* is the size of the matrix and *k* is the size of the pattern mask. We then narrow down the number of kernel patterns used, using the following two criteria: 1) we drop all patterns without adjacent non-zero weights; this is done to keep the semi-structured nature of the kernel patterns; 2) we select the most used kernel pattern by calculating the L2 norm of the kernel using random initiations in the range [-1, 1]. The value of k can range from 1 to 8, which can generate 8 different types of pattern groups. To increase the sparsity level of the model, the number of non-zero weights in a pattern should be low. Prior work [100], [101] on kernel pattern pruning has used 4-entry patterns that consist of 4 non-zero weights in a kernel. But this leads to models with relatively low sparsity and to overcome this issue the authors of these works have utilized connectivity pruning. Due to the drawbacks of connectivity pruning discussed in Section II, we propose to use 3-entry pattern (3EP) and 2-entry pattern (2EP) kernel patterns, which uses 3 and 2 non-zero weights respectively, in our *R-TOSS* framework.

2.4.3 3×3 KERNEL PRUNING

Algorithm 2 shows the pseudocode of the 3×3 kernel pattern pruning using the kernel patterns, illustrated in Figure 13. We start by using the 3×3 parent kernels weights (K_W) from Algorithm 1 as input and initializing a variable (*shape*) to store the shape of the kernel weights (line 1). We also create a pattern dictionary (*kernel_patterns_dict*) consisting of 3EP (Figure 13(a)) and 2EP (Figure 13(b)) patterns (line 3). We then traverse the 3×3 kernels and store the weight matrices of the current 3×3 kernel in the layer as *temp_kernel* (line 5).

ALGORITHM 3: 1×1 KERNEL POOLING

Inputs: *1×1 Kernel Weights (K_W)*

```
FL ← Ø
FL = [w for k in K_W for w in K]
temp_array ← Ø
K'_W ← Ø
for i in range (0, length(FL), 9):
    t1 ← Ø
    t1 = FL [i : i+9]
    if t1.shape[0] == 9 then
        t1 = t1.reshape(3,3)
        L2_norm_array.append(L2norm(t1,2))
        temp_array.append(t1)
    else
        temp_array.append(t1=0)
    Apply 3×3 kernel pruning on temp_array using Algorithm 2
    temp_array = output weight matrices from Algorithm 2
    Reshape temp_array to 1×1 and append to K_W
end
```

Outputs: *pruned 1×1 kernel*

We then initialize an empty list (*L2_dict*) that can store the L2 norm of the *temp_kernel* after applying the kernel pattern from the pattern dictionary. We then iterate through the kernel pattern in the *kernel_patterns_dict* and calculate the L2norm of the kernel after applying the kernel pattern. This L2norm is stored in the *L2_dict* list along with the key of the current pattern from the *kernel_patterns_dict* (lines 7−10). We then find the best kernel pattern for the *temp_kernel* by using the L2norm value from the *L2_dict* and store the index of the kernel pattern in the *bestfit* variable (line 11). The index from *bestfit* is now used as the kernel pattern for the kernel and updated to its original weight matrices K_w (lines 12-14). We then iterate through all the kernels in the parent layer and store this as the kernel mask for the rest of the 3×3 kernels in the parent layer group (l_P) from Algorithm 1.

Once suitable patterns for the parent kernels are found, those patterns are also applied to the corresponding children, by utilizing the convolution mapping. We also apply this pattern matching approach to the 1×1 kernels by performing a 1×1 to 3×3 kernel transformation (see Section 2.4.4).

Since we apply the same kernel mask to all the kernels in a particular group, we can reduce the time taken by the framework to prune the entire model. From experiments, we reduced the total number of patterns required to 21 patterns. Since we have only 21 pre-defined kernel patterns at inference, the kernels with similar patterns are grouped together, which can reduce the overall computational cost and speed up inference.

Table 6 Table showing sensitivity analysis of *R-TOSS* framework in terms of induced sparsity, mAP, and inference time for YOLOv5s and RetinaNet

Entry pattern variant	YOLOv5s				RetinaNet			
	Reduction ratio	***mAP***	***Inference Time (ms)***	***Energy Usage (J)***	***Reduction ratio***	***mAP***	***Inference Time (ms)***	***Energy Usage (J)***
R-TOSS-5EP	1.79×	72.6	11.09	0.97	1.45×	66.09	157.24	14.27
R-TOSS-4EP	2.2×	70.45	10.98	0.91	1.6×	75.8	150.58	13.62
R-TOSS-3EP	2.9×	78.58	6.9	0.478	2.4×	79.45	72.98	6.45
R-TOSS-2EP	4.4×	76.42	6.5	0.454	2.9×	82.9	64.83	5.50

2.4.4 1×1 KERNEL TRANSFORMATION

By performing 1×1 to 3×3 transformation we remove connectivity pruning from kernel pruning. This can ensure we can maintain the accuracy of the model and mitigate losses that arise from connectivity pruning. 1×1 kernel pruning can also speedup inference by grouping similar kernel patterns together.

Algorithm 3 shows the pseudocode for performing 1×1 kernel pruning. We start by using 1×1 kernel weights K_w from the parent layer from Algorithm 1 (*group_list*) as input. We then initialize a list *FL* that is used to store the flattened 1×1 kernel weights from K_w (lines 1-2). Subsequently, a *temp_array* for storing the temporary weight matrices is initialized. We iterate

through the flattened array *FL* and group every 9 weights in the list into 3×3 temporary weight matrices that are stored in *temp_array* (lines 5-11). This process continues till we reach the end of the list or if the values are less than 9. At this point the left-over weights are considered as zero weights and pruned (line 13). We then use Algorithm 2 to perform 3×3 kernel pruning with the temporary 3×3 weight matrices from *temp_array* (line 14). The output matrices from Algorithm 2 are stored back into *temp_array* which is transformed back into 1×1 kernels and appended back into the original 1×1 kernel weights (lines 15-16).

3.

2.5 EXPERIMENTS AND RESULTS

In this section, we evaluate our proposed *R-TOSS* framework on Nvidia RTX 2080Ti and Jetson TX2 platforms in terms of sparsity, mAP, accuracy, and energy usage of the model pruned using our framework and compare it to state-of-the-art pruning techniques.

2.5.1 EXPERIMENTAL SETUP

Our framework is implemented on YOLOv5s, which is a smaller variant of the well-known YOLOv5 with 25 layers and 7.02 Million parameters [37] and RetinaNet that consists of 186 layers and 36.49 million parameters [35]. We implemented ODs with our *R-TOSS* framework using Python and Pytorch and trained them on an NVIDIA RTX 2080Ti GPU [106]. The trained framework is then evaluated using the RTX 2080Ti GPU and also deployed on a Jetson TX2 [107] embedded AI computing device. We use the KITTI automotive dataset [62], with an input image frame size of 640× 640, and a split of 60:40, for training and inference, respectively. The KITTI dataset is a widely used dataset comprised of traffic scenes, making it ideal for AV perception model training. We measure inference times across models in terms of milliseconds (ms) and the mAP with an IoU threshold of 0.5 AP@[.5:.95].

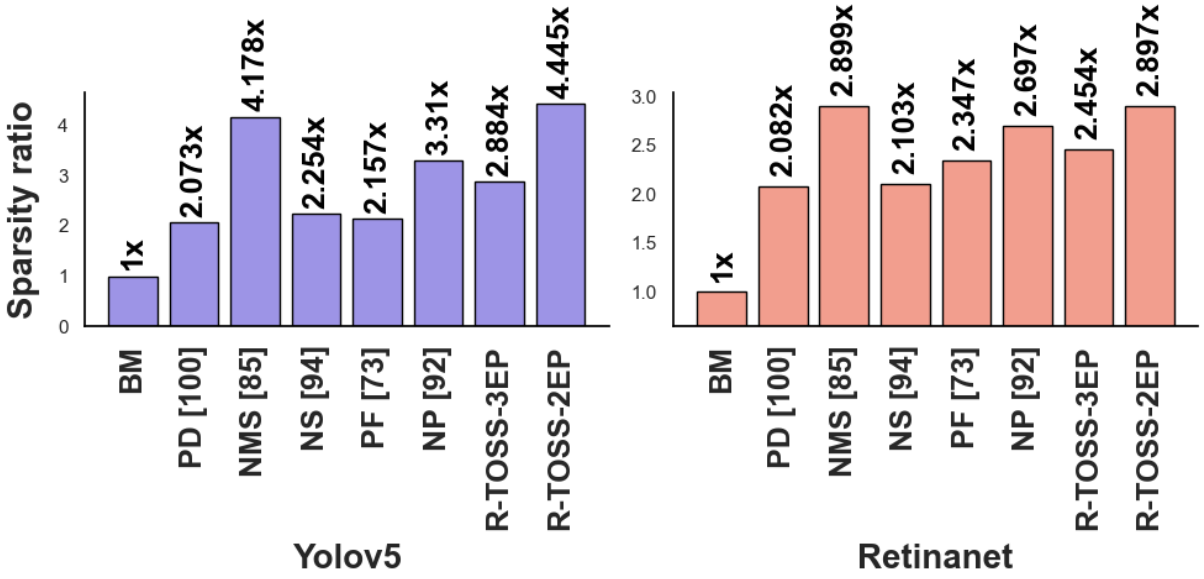


Figure 14 Comparison of sparsity achieved using different frameworks

2.5.2 SENSITIVITY ANALYSIS ON R-TOSS PRUNING FRAMEWORK

We performed a sensitivity analysis study on our *R-TOSS* framework to determine the impact of considering different sizes of kernel patterns. Table 3 shows the results of the study, performed on the RTX 2080Ti. We explored 4-entry patterns (*R-TOSS*-4EP) and 5-entry patterns (*R-TOSS*-5EP), with 4 and 5 nonzero weights in a kernel, respectively, along with our 3EP and 2EP patterns discussed in Section 2.4. From the results it can be seen that while *R-TOSS*-2EP performs better in terms of sparsity induced, inference time, and energy usage on YOLOv5s, it has lesser mAP than when using *R-TOSS*-3EP. We can also observe that 2EP performs better in terms of sparsity induced, mAP, inference time, and energy usage on the RetinaNet model. The performance improvement in terms of inference time and energy usage is due to the higher sparsity achieved of the models. The results indicate that our proposed *R-TOSS*-3EP and *R-TOSS*-2EP pruning frameworks can provide faster and more accurate results than that with the 4EP and 5EP variants

of *R-TOSS*. In the next subsection, we compare the performance of 3EP and 2EP variants of *R-TOSS* with other state-of-the-art pruning frameworks.

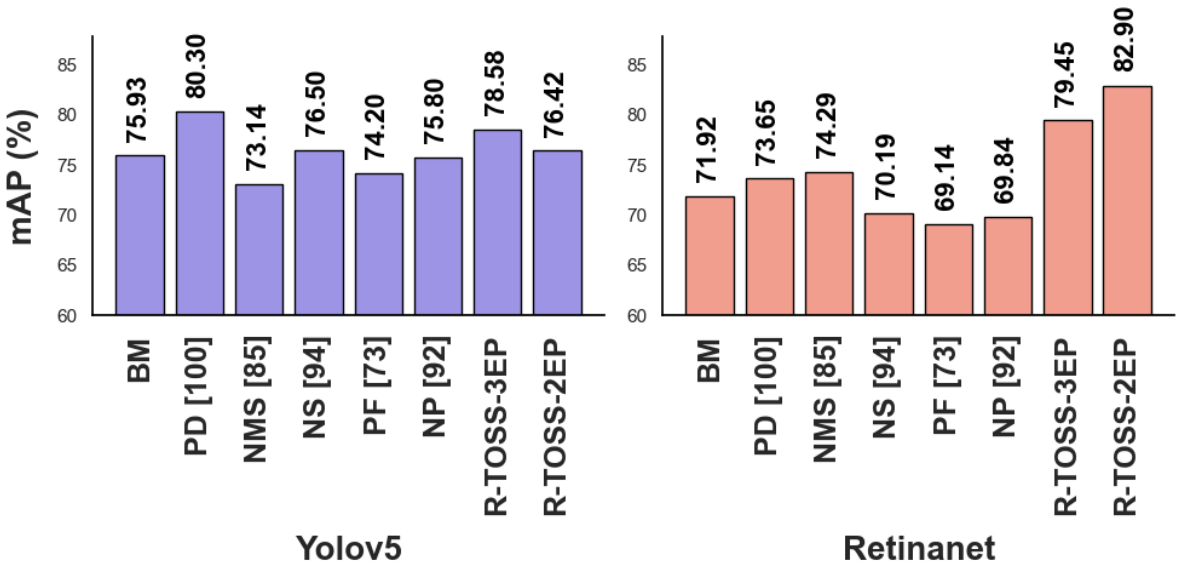


Figure 15 Comparison of mAP achieved using different frameworks

2.5.3 COMPARISON RESULTS WITH OTHER PRUNING FRAMEWORKS

We compared the *R-TOSS*-3EP and *R-TOSS*-2EP with a Base Model (BM) which does not use any pruning, PATDNN (PD) [100] which is a pattern-based pruning technique that uses a 4 entry pattern on 3×3 kernels along with connectivity pruning to increase sparsity, Neural Magic SparseML (NMS) [85] which is an unstructured weight pruning approach that uses the magnitude of the weights in a layer, with the weights below a threshold being pruned, Network Slimming (NS) [94] which uses channel pruning in which a channel is pruned based on a scaling factor for the channel in a layer, Pruning filters (PF) [73] which performs filter granularity weighted pruning, where the total sum of filters weights is calculated and filters below a corresponding threshold are pruned [73], Neural pruning (NP) [92] which uses a combination of filter pruning along with

unstructured weight pruning where L1 norm is used to perform weight pruning and L2 regularization is used to perform filter pruning.

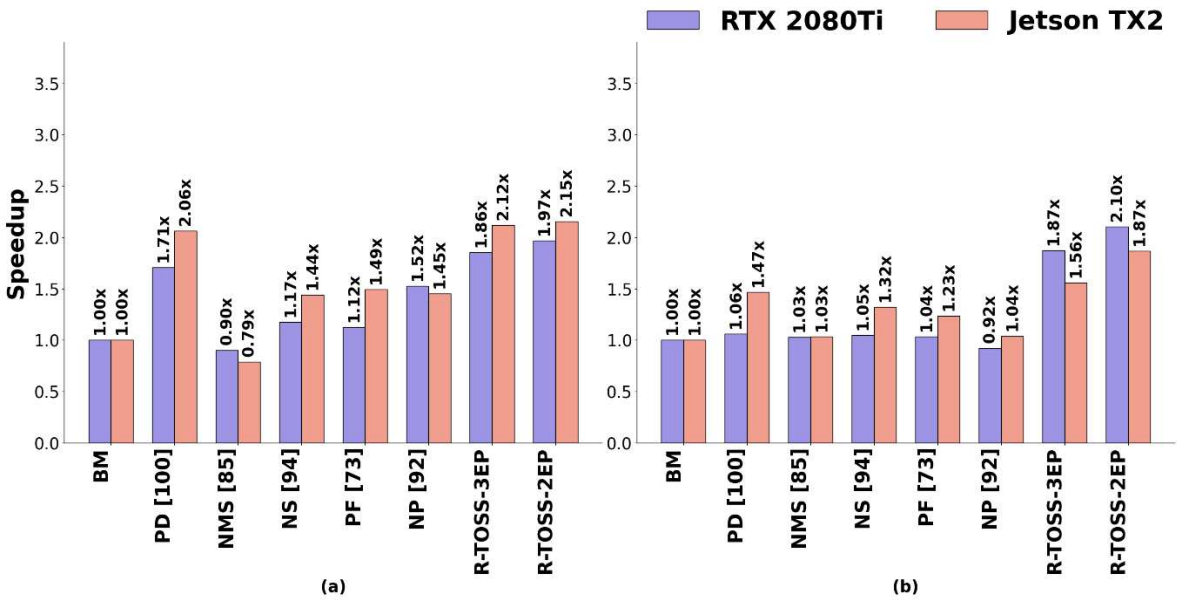


Figure 16 Comparison of speedups achieved in (a) Yolov5 and (b) RetinaNet models after using the pruning frameworks

Figure 14 shows the comparison of the sparsity ratio with other pruning frameworks from prior work, with results normalized to the baseline model BM. It can be observed that our proposed *R-TOSS*-2EP framework achieves very high sparsity across both object detector models. We were able to achieve approximately 2.9× and 4.4× compression on the YOLOv5s model with *R-TOSS*-3EP and *R-TOSS*-2EP, respectively. Similarly, a 2.4× and 2.9× improvement in compression ratio was achieved for RetinaNet with *R-TOSS*-3EP and *R-TOSS*-2EP, respectively. Figure 15 shows the mAP comparison. One can observe that *R-TOSS*-3EP and *R-TOSS*-2EP were able to achieve an mAP of 79.45 % and 82.9% on RetinaNet which is 8.06% and 10.98% better than the best performing framework from prior work (NMS). For YOLOv5s, the *R-TOSS*-3EP variant was outperformed slightly by the PD framework.

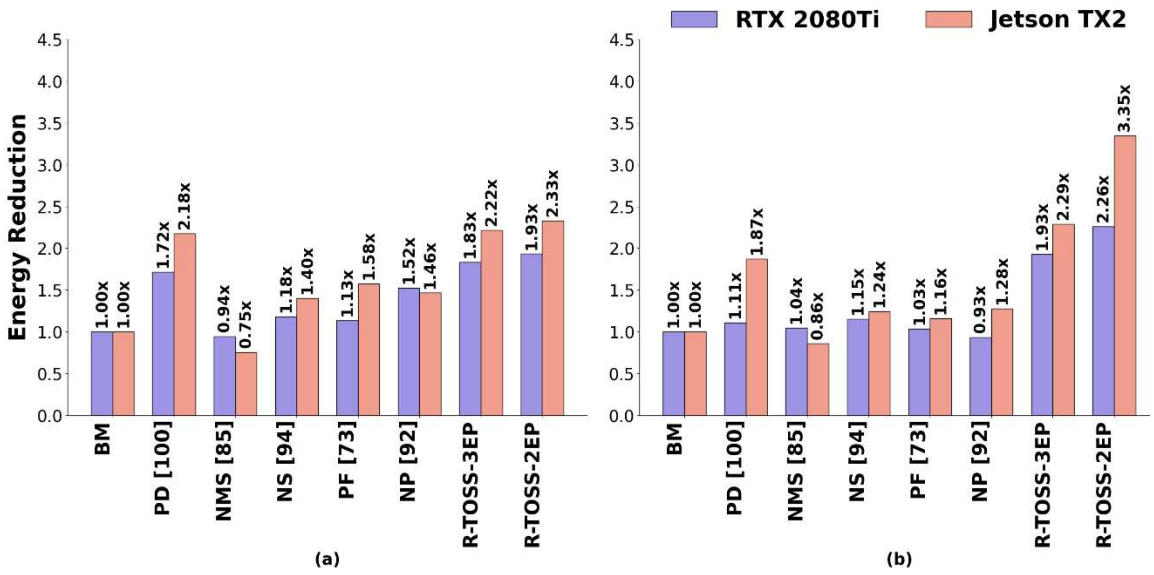
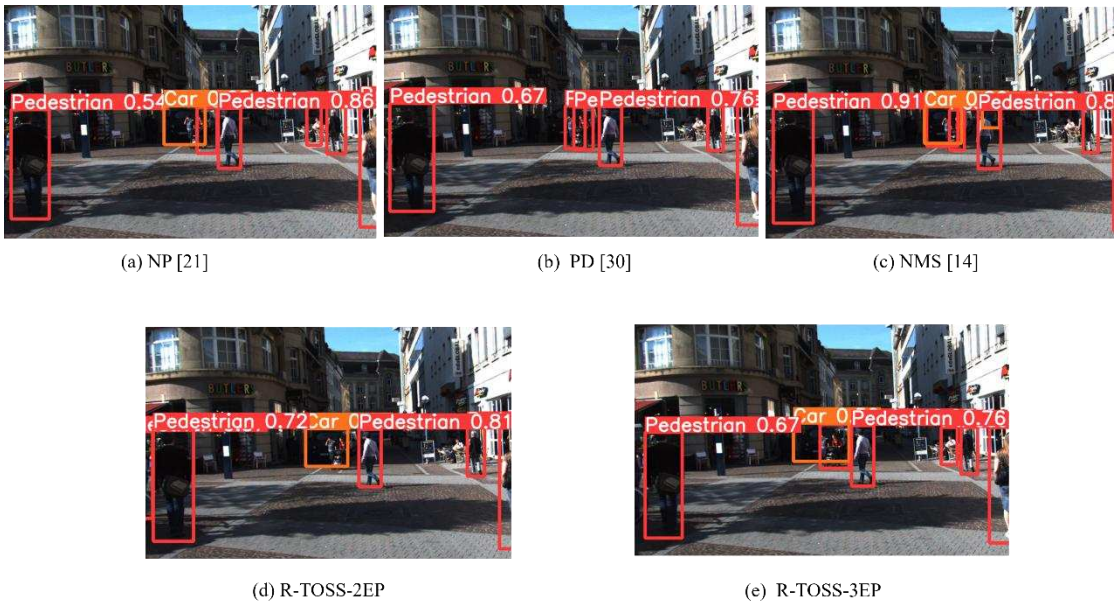


Figure 17 Comparison of reduction in energy usage achieved in (a) Yolov5 and (b) RetinaNet models after using the compression frameworks

Our inference time results in Figure 16 show that RTX 2080 Ti, *R-TOSS*-3EP and *R-TOSS*-2EP were able to achieve a 1.86× and 1.97× speedup in execution time for YOLOv5s and a 1.87× and 2.1× speedup on RetinaNet compared to BM. We also outperform the best performing prior work framework (PD) by 8% and 13.3% for YOLOv5s and 43.3% and 49.6% for RetinaNet with *R-TOSS*-2EP and *R-TOSS*-3EP, respectively. Similarly, on Jetson TX2, *R-TOSS*-3EP and *R-TOSS*-2EP were able to achieve a 2.12× and 2.15× speedup in inference time on YOLOv5s model and 1.56× and 1.87× speedup on RetinaNet compared to BM. *R-TOSS*-3EP and *R-TOSS*-2EP also outperformed PD with 2.6% and 4.27% faster execution time on YOLOv5s and 5.94% and 21.62% on RetinaNet.

The models pruned using our framework also perform better in terms of energy consumption. Figure 17 shows the comparison of reduction in use of energy among various frameworks on both YOLOv5s and RetinaNet. For YOLOv5s, *R-TOSS*-2EP and *R-TOSS*-3EP were able to achieve 45.5% and 48.23% energy reduction over BM and a 6.5 % and 11.2% energy

reduction over PD on RTX 2080Ti; and on the Jetson TX2 they achieved 54.90% and 57.01% reduction over BM and 1.84% and 6.43% reduction over PD. We also observed similar trends on RetinaNet, with 48% and 55.75% reduction of energy usage over BM and 42.46% and 50.97% reduction over PD on RTX 2080 Ti; as well as 56.31% and 70.12% reduction over BM and 18.26% and 44.10% reduction of energy usage over PD on Jetson TX2.



(a) NP [21] (b) PD [30] (c) NMS [14]

(d) R-TOSS-2EP (e) R-TOSS-3EP

Figure 18 Comparison of inference output with other pruning techniques on KITTI automotive dataset using RetinaNet

s

Figure 18 illustrates the performance of different frameworks on a test case from the KITTI dataset. From the results it can be observed that *R-TOSS*-2EP especially retains the ability to detect tiny objects (the car in this example), along with better confidence scores than NP and PD. As AVs rely on fast and accurate inference to take time critical driving decision, *R-TOSS* can help achieve both speed and accuracy while keeping the energy usage lower than that of other state-of-the-art pruning techniques we have compared with.

2.6 CONCLUSION

In this chapter we proposed a new pruning framework (*R-TOSS*) that can outperform several state-of-the-art pruning frameworks in terms of compression ratio and inference time. We were also able to increase the mAP of the ODs compared to the mAP of the baseline models. Overall, our framework achieves significant compression ratios while improving mAP performance on two state-of-the-art object detection models, YOLOv5s and RetinaNet. The proposed framework achieves these results without any compiler optimization or additional hardware requirement. Experimental results on JetsonTX2 show that our pruning framework has a model compression rate of 4.4× on YOLOv5s and 2.89× on RetinaNet while outperforming the original model as well as several state-of-the-art pruning frameworks in terms of accuracy and inference time.

3 UPAQ: A FRAMEWORK FOR REAL-TIME AND ENERGY-EFFICIENT 3D OBJECT DETECTION IN AUTONOMOUS VEHICLES

3.1 INTRODUCTION AND CONTRIBUTION

AVs are essential for enhancing transportation efficiency and significantly reducing crash-related injuries and fatalities [80]. Emerging AVs rely on advanced perception systems—integrating object classification, 3D positioning, and object detection using sensors like LiDARs and cameras. In particular, ODs play a vital role in emerging AVs, being responsible for perceiving an AV's surrounding environment based on captured sensor data and serving as the foundation for the subsequent decision making process. Given their relevance in ensuring safety and executing safety-critical tasks, ODs must deliver high accuracy and achieve real-time inferences within tens of milliseconds [109]. While there have been many advances in ODs accuracy in recent years, these improvements have increased memory footprint and computational overheads on embedded AV platforms [17]. These embedded platforms in AVs also need to process data from multiple on-board systems, V2X communication, and infotainment, which escalates computational demands and reduces power headroom [110]. Modern AVs are increasingly relying on rich 3D data, referred to as pointcloud data, from their sensors and utilizing 3D ODs which can provide depth, size, and location information that 2D ODs cannot. Recent advances have led to complex 3D ODs leveraging sophisticated algorithms for improved accuracy [111]. However, this complexity increases memory usage and computational efficiency compared to the best 2D ODs, reducing real-time performance. To tackle this challenge, advanced model compression techniques have emerged, including pruning, quantization, knowledge distillation, and low-rank factorization

[112]. Among these, pruning stands out for its extensive application in machine learning, increasing parameter sparsity and significantly reducing computational costs [110]. However, conventional pruning methods fall short when considering essential performance metrics such as latency, memory usage, and energy consumption during model execution [113]. This highlights the need for more effective solutions to enhance 3D ODs performance. In this chapter, we present the *UPAQ* framework, designed for efficient 3D ODs compression, via a two-tier compression technique, which leverages kernel pruning and kernel quantization. Central to our approach is a novel model optimization strategy which optimizes the model sparsity and bitwidths while preserving overall accuracy. The novel contributions of our *UPAQ* framework are as follows:

- A model compression approach that retains key feature maps and performs high accuracy real-time 3D object detection with both 3D pointcloud LiDAR data and 3D camera data,
- A kernel transformation approach for quantizing and pruning 1x1 kernels for better generalization of 3D pointcloud features,
- A hybrid approach for mixed precision quantization with semi-structured pruning for improved accuracy retention,
- A model of on-device efficiency of the compressed model to select the best fit quantized kernels for the model,
- A detailed comparison with state-of-the-art ODs pruning and quantization methods, demonstrating the effectiveness of our framework in terms of mean average precision (mAP), latency, energy efficiency, and sparsity.

3.2 RELATED WORK

Before the emergence of 3D ODs, object detection was largely handled through 2D ODs. 2D ODs function by generating bounding boxes on a two-dimensional plane defined by four coordinates: [xmin, ymin, xmax, ymax]. There are two main categories of 2D ODs. Two-stage detectors operate in two steps, first generating region proposals and then classifying those proposals. Examples include Fast R-CNN [29] and Faster R-CNN [30]. These models are extremely resource intensive [114] and have low throughput due to the separate stages involved in object recognition. Single-stage detectors are designed for low latency, functioning with a single feed-forward network. Examples include RetinaNet [35] and YOLOX [39]. However, all of these (and other) 2D ODs models do not possess the ability to perceive depth and cannot utilize the 3D data that a modern sensor suite in AVs can provide.

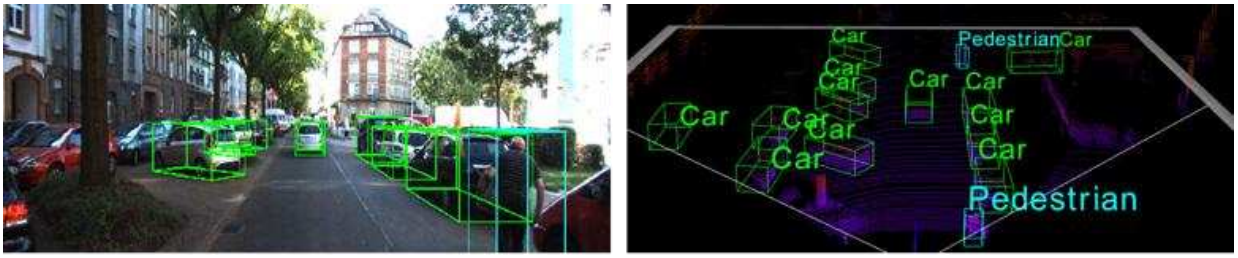


Figure 19 3D object detection example on SMOKE (left) and Pointpillars (right)

3D ODs address the limitations of their 2D counterparts by offering bounding boxes with nine degrees of freedom, incorporating three positional, three dimensional, and three rotational parameters. 3D ODs typically rely on LiDAR pointclouds, RGB cameras, or a combination of these to enhance their ability to interpret their surroundings. *Pointcloud-based 3D ODs models* utilize LiDAR data in a 3D coordinate system, offering high detection accuracy, especially for hard-to-detect objects. These models can process 3D LiDAR data directly (e.g., PointNet [115]) or convert it to 2D (e.g., PointPillars [116]), though the latter may lose features for higher

throughput. *RGB camera-based 3D ODs models* use images for semantic information but lack depth information. So, they typically employ a two-step process of 2D detection followed by 3D bounding box conversion. Examples include Monoflex [117] and SMOKE [118], which are computationally simpler than pointcloud-based models but less accurate, as shown in Figure 19, where SMOKE is unable to detect many objects in the foreground and background. However, even though pointcloud-based 3D ODs models offer higher precision outputs, they have higher computational complexity and a large memory footprint. A few prior efforts have devised lightweight neural networks for ODs, such as SECOND [119] which uses a sparse voxel grid, Focals Conv [120] which uses sparse convolutional layers to speed up inference of pointclouds by focusing on regions with significant data, and VSC [121] which uses a virtual convolution to efficiently process sparse pointcloud data. However, there are still significant challenges to overcome, including the need to more aggressively optimize memory usage and computational efficiency while ensuring high accuracy in diverse and complex environments. The trade-offs between model size and performance (see Table 7) remain critical, particularly as applications demand real-time processing capabilities. Therefore, ongoing research must not only enhance the architecture of sparse networks but also develop advanced techniques for compression, generalization, and robust multimodal integration to further advance the field of real-time 3D ODs.

Table 7 Comparison of 3D ODs model sizes vs execution time

Models	Number of parameters (Millions)	Execution time (ms)
PointPillar [116]	4.8	6.85
SMOKE [118]	19.51	30.65
SECOND [119]	5.3	9.83
Focals Conv [120]	13.70	26.5
VSC [121]	24.5	40.56

Prior works such as Ps and Qs (PQ) [122], Clip-Q [123], LIDAR-PTQ [124], and *R-TOSS* [125] have proposed utilizing quantization and/or unstructured and structured model pruning techniques to compress models and reduce their overheads. The authors in [106] employ Quantization-Aware Training (QAT) with unstructured pruning, but their approach has long training times and reduced model stability. The approach in [123] also combines QAT with unstructured pruning. However, it lacks efficient convergence techniques for balancing accuracy and latency, utilizing a partitioning approach that focuses on only parts of the model without considering overall performance. The approach in [124] uses Post-Training Quantization (PTQ) to convert 32-bit floating-point weights to 8-bit integers, which can enhance the model's inference throughput, but this also reduces accuracy. A recent work [125] employed semi-structured pruning with predefined kernel patterns, known as entry patterns (EPs) to trade-off between structured and unstructured pruning. While promising, the approach lacks the specificity needed to preserve critical features, and the proposed pruning pattern mapping approach leads to suboptimal quantizable weight patterns that compromise inference accuracy. Furthermore, the L2-norm used for selecting the best kernel mask does not adequately account for quantization noise, which further adversely affects performance. In summary, to improve model performance while minimizing latency and maintaining accuracy, it is essential to integrate both pruning and quantization techniques effectively**.** Integrating advanced pruning with quantization methods that account for quantization noise, such as adaptive kernel mask selection, can improve model efficiency and preserve feature extraction accuracy**.**

Our proposed *UPAQ* framework implements such an integrated quantization and pruning approach. Specifically, *UPAQ* supports a semi-structured pattern-based pruning method alongside mixed precision quantization. This strategy increases model sparsity more effectively than

unstructured approaches, preserving critical weights in the kernels for improved accuracy. Additionally, modern point-cloud-based 3D ODs employ pointcloud-to-pseudo-image conversion techniques, such as Pillar Feature Networks [115] [116], to decrease computational complexity during detection. These networks utilize 1×1 convolutional layers to transform and normalize 3D data into a 2D plane, necessitating high precision. Traditional methods that fix the values of these 1×1 convolutional layers during quantization can diminish model accuracy in the earlier layers, leading to overall performance degradation. Our approach addresses this issue by dynamically adjusting the 1×1 kernel weights, thus preserving accuracy during the detection phase while simultaneously creating a sparsity-aware quantized model that effectively reduces the overall footprint.

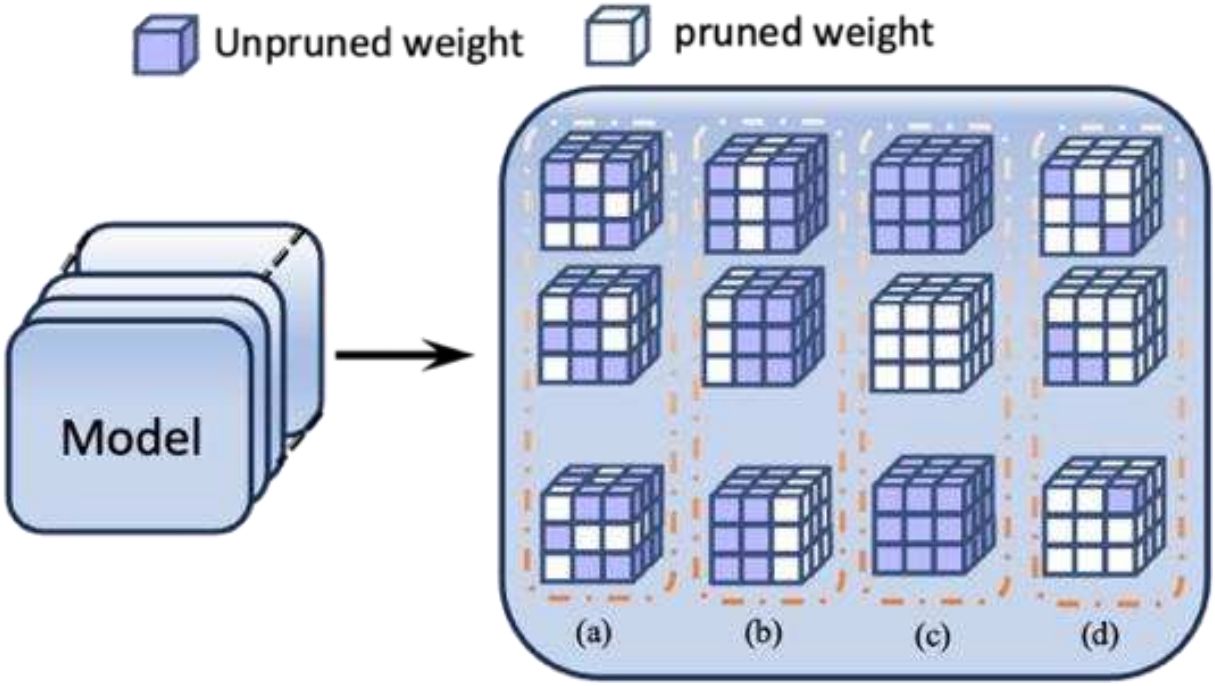


Figure 20 Illustration of different pruning methods (a) unstructured pruning, (b) channel pruning, (c) filter pruning and (d) semi-structured pattern pruning

3.3 MODEL COMPRESSION BACKGROUND

3.3.1 MODEL PRUNING

Pruning is a widely used model compression strategy that promotes sparsity in neural networks through various regularization techniques. This method typically reduces both memory footprint and computational costs while maintaining accuracy. The computational cost of a model can be expressed as:

$$Computational\ cost\ (C) = (L_n \times K_n \times W_n) \qquad (2)$$

where L_n is the number of convolutional layers, K_n denotes the number of kernels in a layer, and W_n represents the number of non-zero weights. As sparsity in the model increases, the computational cost (C) decreases. Recent advances in embedded computing platforms have introduced hardware support for compressing weight matrices during inference, allowing for the omission of zero weights, thus reducing model latency when pruning is employed [125].

Pruning methodologies can be classified into three primary categories: *1) Unstructured Pruning*: This technique selectively prunes weights to minimize model loss while maintaining accuracy (Figure 20(a)). Algorithms in this category include weight magnitude pruning [126], gradient magnitude pruning [127], and second-order derivative pruning [128]. However, these methods can disrupt thread-level parallelism due to load imbalances from varying sparsity levels and may impair memory performance by altering data access patterns, reducing caching efficiency on GPUs, CPUs, and TPUs [128], [129]; *2) Structured Pruning*: This method systematically removes entire channels (Figure 20(b)) or filters (Figure 20(c)) to enhance model sparsity. By creating a uniform weight matrix, filter and channel pruning can significantly reduce multiply-accumulate (MAC) operations compared to unstructured pruning [129]. Structured pruning can be

integrated with acceleration frameworks like TensorRT [96], which can use the uniform pruned structures to optimize hardware acceleration across diverse platforms [130]. However, this approach often decreases model accuracy, as essential weights may be pruned alongside redundant ones; *3) Pattern-based Semi-Structured Pruning*: This approach combines structured and unstructured pruning aspects (Figure 20(d)). It uses kernel masks to selectively retain specific weights, inducing partial sparsity within a kernel. The efficacy of pruned kernels can be evaluated using metrics like the L2-norm. Since kernel patterns are limited to a fixed number of pruned weights, they generally achieve lower sparsity than fully structured or unstructured methods. Connectivity pruning can address this limitation by fully pruning specific kernels [125], [110]. However, pattern pruning often targets kernels of size 3×3 and larger, providing more candidate weights for pruning. Connectivity pruning can end up reducing model accuracy by removing critical weights from kernels. Nonetheless, the semi-structured nature of pattern pruning enables effective hardware parallelism, reducing inference times [125].

3.3.2 MODEL QUANTIZATION

Quantization is a model optimization technique that reduces memory footprint and computational costs by converting weights (and optional activations) from higher floating-point precision to lower precision. With advancements in hardware and software, many platforms now support precision levels as low as 1-bit integers.

Approaches for model quantization can be divided into two types based on when quantization occurs during model development: 1) Quantization Aware Training (QAT) involves adding quantization and de-quantization nodes to a fully trained model and retraining it for a set number of epochs. The model calculates the scale factor and simulates quantization loss, integrating it into the overall loss function through fine-tuning, which enhances robustness for

subsequent Post-Training Quantization [131]; 2) Post-Training Quantization (PTQ) quantizes a fully converged model using various algorithms and precision levels. Quantization can also be classified into two categories based on how parameters are changed [132]: 1) Integer quantization: Here a calibration dataset is used to convert 32-bit floating-point parameters to fixed-point integers (e.g., 8-bit) by calculating the minimum and maximum values of model parameters using the calibration dataset; 2) Floating point quantization: This method reduces precision from 32-bit to lower bitwidth (e.g., 16-bit) floating-point tensors, prioritizing model accuracy compared to integer quantization. In most approaches, a common practice is to quantize all layers in a model to the same bit-width. However, for many models there is a distinct difference in sensitivity to quantization from layer to layer. Mixed precision quantization addresses this issue by keeping more sensitive layers at higher precision while maintaining the rest of the model in lower bits, effectively improving the performance-efficiency trade-off [132].

3.4 *UPAQ* FRAMEWORK

In this section, we describe our novel 3D ODs model compression framework and provide a detailed algorithmic description of our kernel pruning and quantization techniques. Our compression framework, *UPAQ* (Figure 21), combines a semi-structured pattern pruning scheme with mixed precision quantization, while incorporating various optimizations to reduce computational costs for 3D ODs models. The *UPAQ* framework has three stages: pre-processing, pattern generation, and compression. These are discussed next.

3.4.1 PRE-PROCESSING STAGE

In this first stage, we begin by calculating the computational graph of the pretrained model M and utilizing the DFS algorithm to determine the computation paths or connected layers. A root

layer may be shared among multiple leaf layers, which we leverage to significantly lower the computational cost associated with optimizing the model. *UPAQ* applies key optimizations to the root layers, so that they are reflected in the leaf layers through forward passes, rather than managing individual layers. The preprocessing stage identifies the root layer-leaf layer subsets within the computation graph so that the optimization steps can be performed with lowered computation cost in the later compression stage (Subsection 3.4.3).

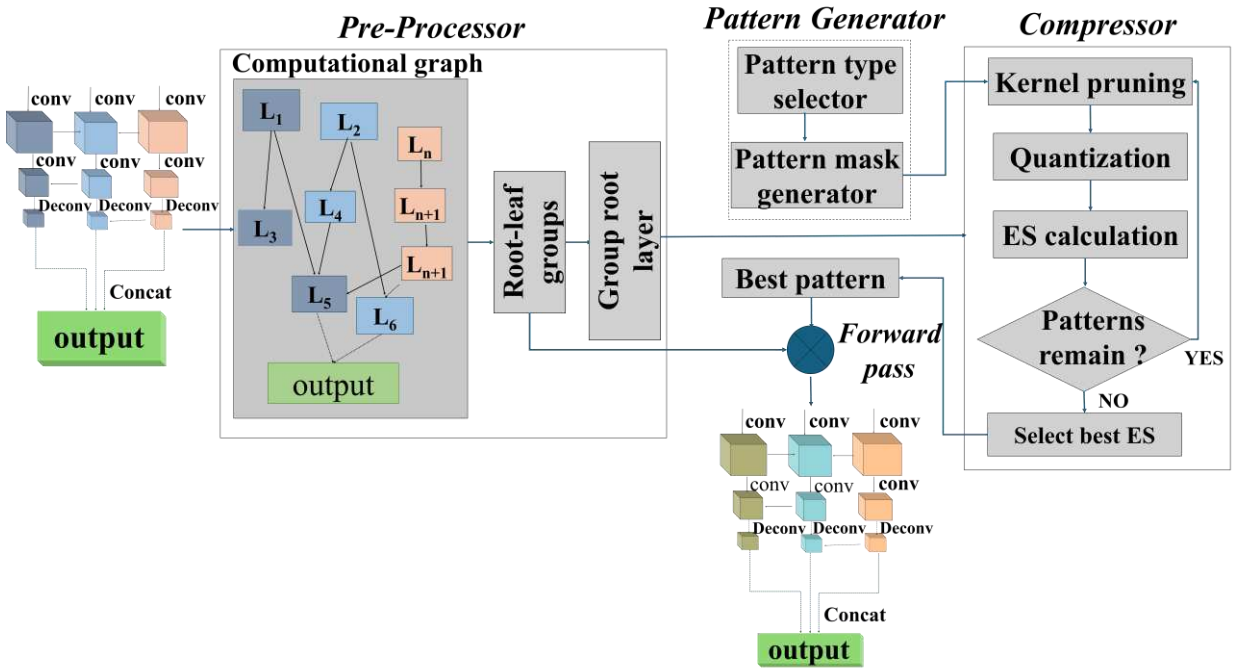


Figure 21 An overview of the proposed UPAQ optimization framework

Algorithm 4 outlines the pseudocode for the preprocessing stage. We start by computing the computational graph G of the model through backpropagation (line 1). This graph G is used to execute DFS via the find_root function by traversing model layers l and identifying the root for each current layer. A layer can be classified as the root layer when it does not have any other layer designated as its root (lr), indicating that it becomes its own root (line 4). This root layer is incorporated into the list of groups initialized as groups_int (line 2). If a layer is identified as

belonging to an existing group, it adopts the corresponding root layer (lr) and is added to that group (lines 5-6). Each root layer (lr) can have multiple associated layers, but each of these layers can only be linked to one root layer. This process continues until all layers are categorized into a group. The layers within each group share kernel properties due to their interconnected channels, allowing them to adhere to the same optimization pattern.

ALGORITHM 4: PREPROCESSING

Inputs: *Baseline pretrained model (M)*
Output: *A list of root-leaf layer groups (group)*

```
1  G ← compute_graph(M)
2  groups_int ← {}
3  for l in M:
4        l_r ← find_root (G, l)
5        if : [l_r] in groups_int then: groups_int [l_r] ← [M [l]];
6        else: groups_int [ l_r ] ← 0; groups_int [ l_r ] ← [M [l]];
7  end
```

3.4.2 PATTERN GENERATION STAGE

The pattern generator algorithm generates a random pattern of non-zero elements within a k×k kernel, utilizing one of four potential arrangements: main-diagonal, anti-diagonal, row, or column. This combined with our compression stage algorithm (Subsection 3.4.3) ensures that we can obtain the best possible compression for our optimized model, compared to relying on a dictionary of patterns.

Algorithm 5 outlines the pattern generation approach. First, we randomly select one of the patterns from the predefined list (line 1). This selection will dictate how we place the non-zero elements in the kernel. We then initialize positions an empty array which will store the coordinates of the non-zero weights in a kernel (line 2). If the chosen pattern is the main_diagonal, we calculate

the position of the non-zero elements as (i, i) for the range of 0 to min(n-d)-1 where n and d are the number of non-zero weights and dimension of the kernel (line 3-4).

ALGORITHM 5: PATTERN GENERATOR

```
    Inputs: number of non-zero weights (n), dimension of kernel (d)
    Outputs: Kernel pattern
1   pattern_type ← random choice from ['main_diagonal', 'anti_diagonal', 'row', 'column']
2   positions ← []
3   if pattern_type = 'main_diagonal' then
4       positions ← [(i, i) for i in (0, min(n, d)]]
5   else if pattern_type = 'anti_diagonal' then
6       positions ← [(i, d -i -1) for i in (0, min(n, d)]]
7   else if pattern_type == 'row' then
8       row ← random choice from (0, d]
9       start_col ← random choice from (0, d-n]
10      positions ← [(row, start_col + i) for i in (0, n]]
11  else if pattern_type == 'column' then
12      col ← random choice from (0, d]
13      start_row← random choice from (0, d-n]
14      positions ← [(start_row + i, col) for i in (0, n]]
15  end
```

When the selected pattern is anti-diagonal, we place the non-zero elements at (i, d-i-1) for the range of 0 to min(n-d)-1 (line 5-6). In the case where a row pattern is selected, we choose a random row of range (0, d-n]. We then choose a starting column (start_col) along which the non-zero weights are placed which is then used to select the position of the elements using (row, start_col + i) for the range of (0, n] (line 7-10). For instance, if we randomly select column 0 as start_col, and if three non-zero elements are needed, we fill all three columns in the selected row, resulting in positions (row, 0), (row, 1), and (row, 2). Finally, if we select a column pattern we choose a random column of range (0, d]. We then choose a starting row (start_row) along which the non-zero weights are placed which is then used to select the position of the elements using (start_row + i, col) for the range of (0, n] (line 11-14). For example, if three non-zero elements are required, we fill all three rows in the chosen column, resulting in positions (start_row, col),

(start_row + 1, col), and (start_row + 2, col). The output of this algorithm is a set of coordinates representing locations where the non-zero elements will be placed in the kernel. This pattern selection ensures that the kernel mask is varied and avoids symmetry along diagonal/row/column depending on the pattern_type selected.

ALGORITHM 6: COMPRESSION STAGE FRAMEWORK

Inputs: *Pretrained model (M) , group_int*
Outputs: *compressed model (M_C)*
1 $M_C \leftarrow$ *deepcopy(M)*
2 **for** *l in* M_c*:*
3 **if** *l in group_init[*l_r*].keys()* **then**
4 *weights = weights in l*
5 **for** K_W *in weights:*
6 ***shape*** $\leftarrow$ K_W*.shape*
7 **if** *shape[-1] > 1 then*
8 *Call Algorithm 4 to perform k×k kernel compression*
9 *Apply the same compression pattern to all kernels in leafnode*
10 **else*:***
11 *Call Algorithm 5 to perform 1×1 kernel compression*
12 *Apply the same compression pattern to all kernels in leafnode*

3.4.3 COMPRESSION STAGE

In this final stage, we utilize a combination of pruning and quantization to minimize the model footprint. The process includes: *1)* executing semi-structured pattern pruning; *2)* quantizing the weights of the pruned model; and *3)* performance modeling of the compressed model operation on-device, to evaluate the impact on on-device model performance. This is an optimization stage which ensures the best possible model compression while optimizing model performance in terms of accuracy, latency, and energy consumption.Algorithm 6 outlines our overall framework for compressing a pretrained model *M* by pruning and quantizing its kernels to produce a compressed model M_C. The process begins by creating M_C as a deep copy of *M* (**line 1**). Using a deep copy [133] is critical here because it allows us to modify the structure and weights of M_C independently

of M, preserving the original model for comparisons. This separation is essential in model compression tasks to evaluate the compressed model effectiveness without altering the baseline model structure and performance. We then iterate through each layer l in M_C (**line 2**), checking if the layer is part of the root layer of the *group_init* (**line 3**). For each layer that qualifies, it retrieves the weights (**line 4**) and examines each kernel K_w (**line 5**). We then check the dimension of the kernel. If the kernel shape is not 1×1 (**line 7**), it applies Algorithm 7 to perform $k\times k$ kernel compression (**line 8**) and replicates the same compression pattern for all kernels in the corresponding leaf node (**line 9**). Conversely, if the kernel is 1×1 (**line 10**), it uses Algorithm 8 for compression (**line 11**) and applies the compression pattern to all leaf node kernels (**line 12**). This method ensures an efficient compression strategy tailored to the structure of the model's kernels.

ALGORITHM 7: K×K KERNEL COMPRESSION

```
        Inputs: k×k Kernel Weights (K_W)
        Outputs: compressed k×k kernels, bestfit_pattern
1       best_E_S ← Ø
2       bestfit_kernel ← Ø
        quant_bit ← array of quantization range
3       temp_kernel ← create a zeros array of shape K_W
4       positions [] ← pattern_generator(n, K_W.shape[-1])
5       for p in positions:
6         temp_kernel [p] = temp_array [p]
7           temp_kernel [p] = K_W [p]
8           for q in quant_bit:
9             compressed_kernel, sqnr =mp_quantizer (temp_kernel, q)
10            K_W = compressed_kernel
11            E_S ← calculate_ES(Mc, sqnr)
12            if E_S > best_ E_S:
13              best_E_S = E_S
14              bestfit_kernel [l_r]= K_W
15          end
16      end
```

3.4.3.1 KERNEL COMPRESSION

Our framework above (Algorithm 6) requires efficient compression of k×k and 1×1 kernels, the approach for which is discussed next.

ALGORITHM 8: 1×1 KERNEL COMPRESSION

Inputs: *1×1 Kernel Weights (K_W)*
Outputs: *compressed 1×1 kernel, bestfit_pattern*

```
1   fl = [w for w in row for row in K_W] # flatten list of 1×1 kernel weights
2   best_E_S ← Ø
3   temp_array← Ø
4   bestfit_kernel ← Ø
5   quant_bit ← array of quantization ranges
6   temp_kernel ← create a zeros array of shape K_W
7   for i in range (0, len (fl), k):
8       t1 = fl [i : i+k]
9       if t1.shape[0] == k then
10          t1 = t1.reshape(k,k)
11          temp_array.append(t1)
12      else: temp_array.append(t1=0);
13  end
14  positions [] ← pattern_generator(n, K_W.shape[-1])
15  for p in positions:
16      temp_kernel [p] = temp_array [p]
17      for q in quant_bit
18        compressed_kernel, sqnr = mp_quantizer(temp_kernel, q)
19        K_W = Flatten(compressed_kernel)
20        E_S ← calculate_E_S (M_c , sqnr)
21        if E_S > best_E_S:
22          best_E_S = E_S
23          bestfit_kernel [l_r]= K_W
24      end
25  end
```

The k×k kernel compression algorithm (Algorithm 7) performs kernel-wise compression through pruning and quantization. It makes use of a mixed precision quantizer (Algorithm 6) and calculates an efficiency score (ES from eq. (3); discussed later) after applying the compressed kernel back to the model. In Algorithm 4, we set several variables: best_ES, bestfit_kernel, and temp_kernel (lines 1-3). Patterns are generated using pattern_generator (Algorithm 2; line 4) to

induce sparsity in the kernels. Next, we iterate through the kernel weights (KW) of the root layer and utilize the positions (rows and cols) of the non-zero kernels identified from the generated patterns (lines 5-7). These row and column positions are used to establish the locations of the non-zero kernels, which are then processed by mp_quantizer (Algorithm 9) for quantization with different quantization bitwidths (q). The compressed kernel is applied back to the kernel weights (lines 8-10). After this, the Mc is used to calculate the ES (from eq. (3); discussed later), iterating through all generated patterns (line 11). The pattern that results in the highest ES is designated as the best kernel (bestfit_pattern) for the root layer (lines 11-14). This process is repeated for all k×k kernels in the root layer of the groups.

For *1×1* kernels, even though these are abundant in most modern deep neural networks, their compression is often overlooked. To compress these kernels, we use a *1×1* to *k×k* transformation algorithm that enables grouped pruning and quantization.

Algorithm 8 presents the pseudocode for this transformation and compression process. First, we reshape the *1×1* kernel (K_W) from the root layer, as described in Algorithm 4, flattening them and storing the result in *flatten_list* (**line 1**). We then iterate through *flatten_list*, grouping the values into sets of *k* weights, which are reshaped into a *k×k* weight matrix and stored in *temp_array* (**lines 7-13**). Next, we use our *pattern_generator* to randomly generate a pattern as per the number of non-zero elements and the resized kernel size (Algorithm 5; **line 14**). Utilizing the positions (*rows, cols*) of non-zero kernels identified from the generated patterns, we compress and quantize the *temp_kernels* in *temp_array* (**lines 15-25**). For quantizing *temp_kernel*, we use the *mp_quantizer* (Algorithm 6), which considers various bit-size alternatives (**line 18**). The modified kernel is then applied back to the kernel weights (K_W) by flattening the weights back to a *1×1* format (**line 19**). Afterwards, we calculate the efficiency score E_s (eq. (2); discussed later), for all

the generated kernels (**line 20**). The kernel with the highest E_S is selected as the best kernel (*bestfit_pattern*) for the root layer (**lines 21-23**).

For both 1×1 and $k\times k$ kernels, given that the layers in the root group have coupled channels, the *bestfit_pattern* is employed to apply quantization to the leaf layers within the root group. The process of pruning, quantization, and kernel selection creates a large search space due to the different quantization bits and kernel combinations. To reduce the computational complexity of this exploration, we compress only the group root layers from *group_init*, which significantly limits the number of potential combinations to explore. By focusing on the root layers, we can more efficiently prune and quantize the kernels, reducing the computational burden of testing all layers and configurations. After applying the best pattern to the root layers, these optimized kernels are also applied to the subsequent leaf layers, making the algorithm more computationally feasible while still achieving effective compression and quantization.

ALGORITHM 9: MP_QUANTIZER

Inputs: *temp_kernel*
Outputs: *compressed_kernel, sqnr*

1 $\boldsymbol{x} = temp\_kernel$
2 $\alpha_x = max(|min(x)|,|max(x)|)$
3 $max\_value = 2^{(quant\_bit - 1)} - 1$ *# quant_bit is the integer bit quantization*
4 $min\_value = -(2^{(quant\_bit - 1)} - 1)$
5 $scale = \alpha_x / max\_value$
6 $x_q = round(x/scale)$
7 $x_q = clip(x_q, min\_val, max\_val)$
8 $sqnr = var(x)/var(x - x_q)$

3.4.3.2 MIXED PRECISION QUANTIZATION

We perform symmetric quantization, where we map the floating-point representation of each kernel weight to an equivalent integer space, such that both the real (floating point) and integer

space is centered around 0. This approach enhances memory efficiency, especially in pruned models, by treating both positive and negative values equally, thus speeding up inference. Symmetric quantization can also result in higher Signal-to-Quantization-Noise Ratio (SQNR), preserving accuracy while minimizing quantization error. Symmetric quantization enables faster, more power-efficient inference by leveraging fixed-point arithmetic, improving throughput and latency in embedded platforms [134].

Table 8 Comparison of *UPAQ* with base (uncompressed) model and state-of-the-art compression frameworks for the Pointpillars and SMOKE 3D ODs.

Models	Metrics		Frameworks						
			Base Model	Ps&Qs [122]	CLIP-Q [123]	*R-TOSS* [125]	LIDAR-PTQ [124]	***UPAQ* (LCK)**	***UPAQ* (HCK)**
PointPillars	***Compression***		1×	1.89×	1.84×	4.07×	3.25×	**4.92×**	**5.62×**
	mAP		78.96	83.67	79.68	85.26	78.90	**86.15**	**84.25**
	Inference time (ms)	***RTX 4080***	5.72	5.17	5.26	5.69	4.25	**2.37**	**1.70**
		Jetson Orin	35.98	32.061	35.07	35.94	29.65	**19.96**	**18.23**
	Energy Usage (J)	***RTX 4080***	0.875	0.658	0.716	0.871	0.567	**0.371**	**0.327**
		Jetson Orin	0.863	0.782	0.841	0.862	0.711	**0.472**	**0.417**
SMOKE	***Compression***		1×	1.95×	1.84×	4.25×	3.57×	**4.23×**	**5.13×**
	mAP		29.85	31.03	30.45	32.56	30.23	**36.65**	**35.49**
	Inference time (ms)	***RTX 4080***	28.36	23.72	25.48	24.98	12.75	**9.67**	**8.23**
		Jetson Orin	127.48	93.65	87.28	98.87	86.27	**71.35**	**68.45**
	Energy Usage (J)	***RTX 4080***	8.95	7.79	8.63	4.37	4.79	**3.21**	**2.83**
		Jetson Orin	25.85	19.21	17.87	20.84	18.25	**15.62**	**13.80**

Algorithm 9 presents the pseudocode for our quantization algorithm (*mp_quantizer*). We begin by copying the pruned *temp_kernel* to $\boldsymbol{x}$ and then calculating the scaling factor (α_x) for the kernel, which enables the mapping of continuous data to discrete representations while maintaining the integrity of the original information (**lines 1-2**).The scaling factor is the maximum absolute value of either the minimum or maximum of x, ensuring that the quantization scale will be appropriate for the range of values in the kernel. We calculate the maximum quantization value (*max_value*) and the minimum quantization value (*min_value*) based on the desired number of

quantization bits (*quant_bit*) (**line 3-4**). We then compute the *scale* as the ratio of α_x to the *max_value*, which determines the factor by which the kernel values will be scaled during quantization (**line 5**). The scale is then used to compute the quantized weight (x_q) by dividing the original values by the scale and rounding them to the nearest integer (**line 6**). We then apply a clipping operation to ensure that the quantized values remain within the range of (*min_value, max_value*) (**line 7).** Finally, the error between the pruned weight (x) and the quantized weight (x_q) is used to compute the SQNR, which provides a measure of the quantization error relative to the original kernel (**line 8**). The final outputs of the algorithm are the quantized kernel weights and the SQNR.

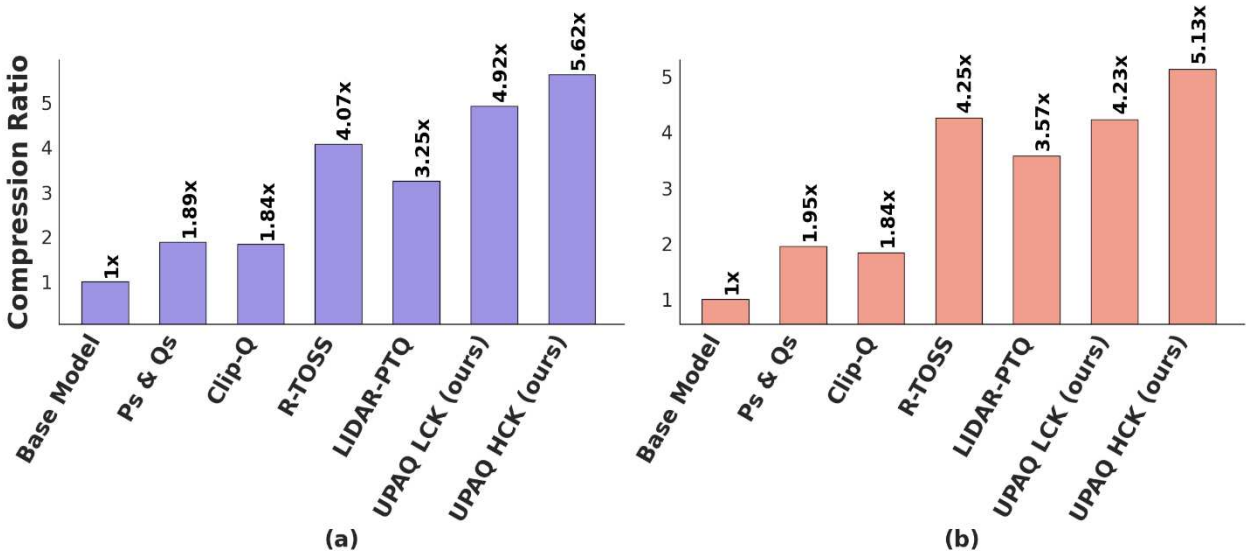


Figure 22 Compression ratio achieved in (a) PointPillars and (b) SMOKE across various compression frameworks

Our kernel compression algorithms (Algorithms 7 and 8) employ the *mp_quantizer* (Algorithm 9) along with iterative search through the *quant_bit* array to find the best E_s, to implement a mixed-precision quantizer. Through this approach, we can allow lower precision (e.g., 4 bits) for less

significant kernels and higher precision (e.g., 16 bits) for more important kernels, to balance model footprint and accuracy.

3.4.3.3 EFFICIENCY SCORE

We compute the efficiency score (E_S) of the model after updating the compressed kernel weight to the model M_c as:

$$E_s = \alpha.sqnr + \beta.\left(\frac{1}{Latency}\right) + \gamma.\left(\frac{1}{Energy}\right) \qquad (3)$$

where α, β, γ are weights between [0,1] that determine the importance of each component in the efficiency score. We then calculate the on-device latency and energy of the model and use these values and calculate the E_S of the model.

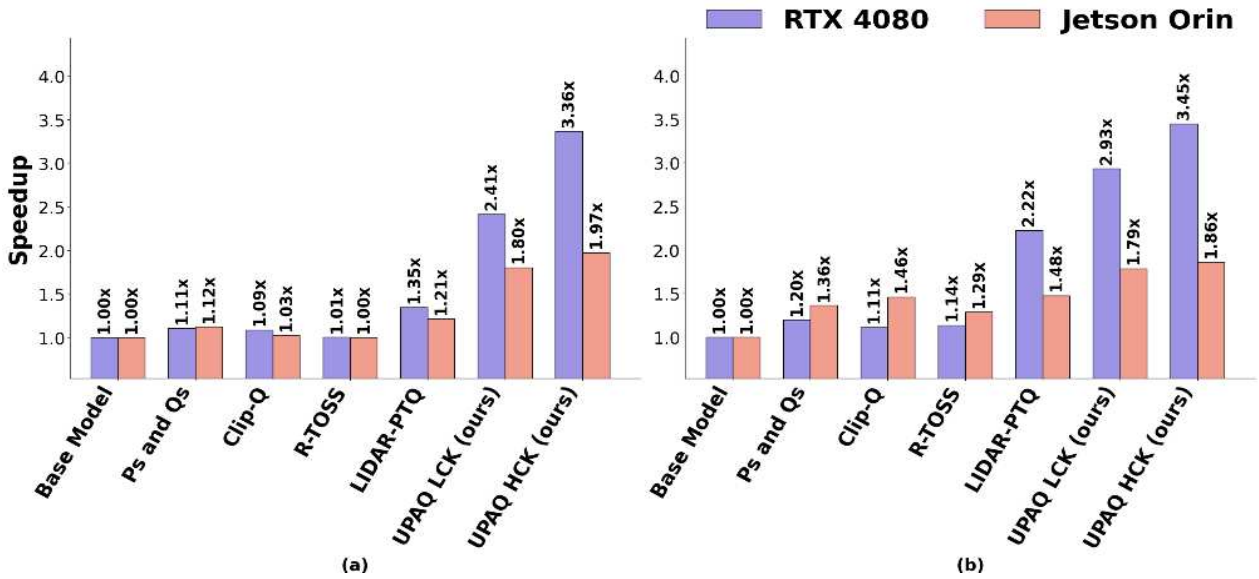


Figure 23 Comparison of speedups achieved in (a) PointPillars and (b) SMOKE models after using the compression frameworks

3.5 EXPERIMENTAL RESULTS

In this section, we present the results of prototyping and implementing our proposed *UPAQ* framework on the Jetson Orin Nano embedded platform and an Nvidia RTX 4080 workstation. We also contrast *UPAQ* with state-of-the-art compression techniques for 3D ODs.

3.5.1 EXPERIMENTAL SETUP

Our framework is evaluated on two state-of-the-art pretrained 3D ODs: 1) *PointPillars*, which uses pointcloud LiDAR data with 4.8 million parameters and an inference time of 35.98ms for the uncompressed model on Jetson Orin; and 2) *SMOKE*, which uses image-based input with 2D to 3D uplifting, consisting of 19.51 million parameters and 173 layers, with an inference time of 127.48ms for the uncompressed model on Jetson Orin. We implemented and tested the framework using PyTorch and TensorRT, on an Nvidia RTX 4080 workstation, and then deployed the model on the Jetson Orin. We calculate the power consumption of these models using NVpower tool [135]. The evaluation metrics include: 1) compression ratio; 2) mAP; 3) inference time; and 4) energy usage. We use the KITTI automotive dataset [62], split 80:10:10 for training, validation, and testing of both LiDAR pointcloud and RGB images.

We evaluate two variants of the *UPAQ* framework: *1) UPAQ* (HCK): which is biased towards higher compression, with fewer non-zero weights per kernel (e.g., 2 non-zero values for a 3×3 kernel) and more aggressive quantization with lower quantization bitwidths (e.g., a mix of 4 and 8 bits); and 2) *UPAQ* (LCK): which is biased towards greater accuracy, with more non-zero weights than HCK (e.g., 3 non-zero values for a 3×3 kernel), and less aggressive quantization (e.g., a mix of 8 and 16 bits). The quantization bits (*quant_bits*) considered for experiments vary from 4 to 16 and we also set the weights in E_s to be $\alpha=0.3, \beta=0.4, \gamma=0.3$ so that we give higher significance to minimizing model latency in our optimizations.

3.5.2 EVALUATION RESULTS OF UPAQ COMPRESSION FRAMEWORK

We compared our *UPAQ* framework with the uncompressed Base Model (BM) and four state-of-the-art approaches. These include Ps&Qs (PQ) [122] which uses quantization-aware pruning with iterative pruning and pre-layer quantization using the same number of quantization bits. We also consider Clip-Q [123], which applies clipping, partitioning, and quantization. In this method, clipped weights are pruned, and non-clipped weights are quantized. We also consider *R-TOSS* [125] which applies entry-pattern based semi-structured pruning. Lastly, we consider Lidar-PTQ [124] that uses PTQ with max-min calibration and adaptive rounding for weight quantization in 3D ODs. Table 2 summarizes our evaluation results which will be discussed in more detail.

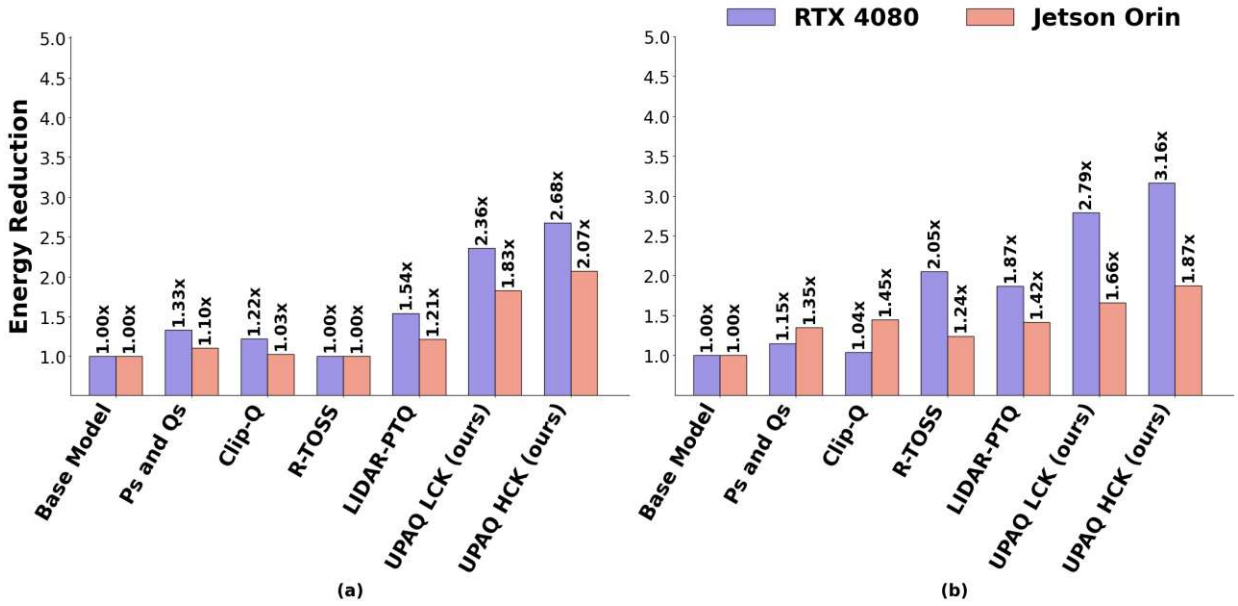


Figure 24 Comparison of reduction in energy usage achieved in (a) PointPillars and (b) SMOKE models after using the compression frameworks

In terms of accuracy, for PointPillars, *UPAQ* (LCK) achieves the best mAP of 86.15, surpassing Ps&Qs (83.67) and *R-TOSS* (85.26), while *UPAQ* (HCK) is close with an mAP of 84.25. For the SMOKE model, *UPAQ* (LCK) also delivers the highest mAP of 36.65.

In terms of compression (Figure 22), for the PointPillars model, *UPAQ* (HCK) achieves the highest compression ratio of 5.62×, outperforming Ps&Qs (1.89×), CLIP-Q (1.84×), LIDAR-PTQ (3.25×) and *R-TOSS* (4.07×). For the SMOKE model, *UPAQ* (HCK) also has the highest compression ratio of 5.13×, outperforming all other frameworks.

In terms of inference speedup on the Jetson Orin (Figure 23), for PointPillars, *UPAQ* (HCK) achieves an inference time of 18.23ms, which is 1.97× faster than the base model (35.98ms) and faster than all other frameworks. *UPAQ* (LCK) reduces inference time to 19.96ms, which is 1.81× faster than the base model. For SMOKE, *UPAQ* (HCK) delivers an inference time of 68.45ms, which is 1.86× faster than the base model and faster than all other frameworks. *UPAQ* (LCK) also improves on the base model by 1.78× (an inference time of 71.35ms).

In terms of energy usage on the Jetson Orin (Figure. 24), *UPAQ* (HCK) for PointPillars uses 0.417 J, which is 2.07× lower than the base model (0.863 J) and more efficient than Ps&Qs (0.782 J), CLIP-Q (0.841 J), LIDAR-PTQ (0.711 J) and *R-TOSS* (0.862 J). *UPAQ* (LCK) uses 0.472 J, 1.83× more efficient than the base model. For SMOKE, *UPAQ* (HCK) uses 15.62 J, which is 1.87× lower than the base (25.85 J) and more efficient than the other frameworks. *UPAQ* (LCK) reduces energy to 13.80 J, which is 1.66× more efficient than the base model.

Lastly, in Figure. 25, we compare object detection predictions across various frameworks using 3D pointcloud data from the KITTI dataset. In the figures, blue bounding boxes indicate the ground truth positions of cars, while red boxes show each frameworks predictions. We have selected three of the best performing models in terms of prediction accuracy, *R-TOSS*, *UPAQ* (HCK), and *UPAQ* (LCK), to be contrasted against the Base Model performance of PointPillars. For *R-TOSS* (85.26 mAP), even though it predicts all cars in the scene, it has misalignments in the results, which can lead to incorrect 3D positioning in driver assitance applications. *UPAQ* (LCK)

achieves higher accuracy (86.15 mAP), with bounding boxes closely aligned with the ground truth and no extraneous predictions. *UPAQ* (HCK) also performs well, with minimal deviations from the true positions, closely matching *UPAQ* (LCK) in precision (84.25 mAP). Overall, our *UPAQ* frameworks, particularly *UPAQ* (LCK), demonstrate superior accuracy and reliability in 3D object detection, over other frameworks.

In summary, if accuracy is the priority, we recommend the LCK configuration for its superior mean Average Precision (mAP), despite slightly lower compression and speedup compared to HCK. For maximizing compression, inference speed, and energy efficiency, especially in resource-constrained environments, HCK is the better choice, offering higher compression, faster inference, and lower energy usage, at the cost of a slight reduction in accuracy.

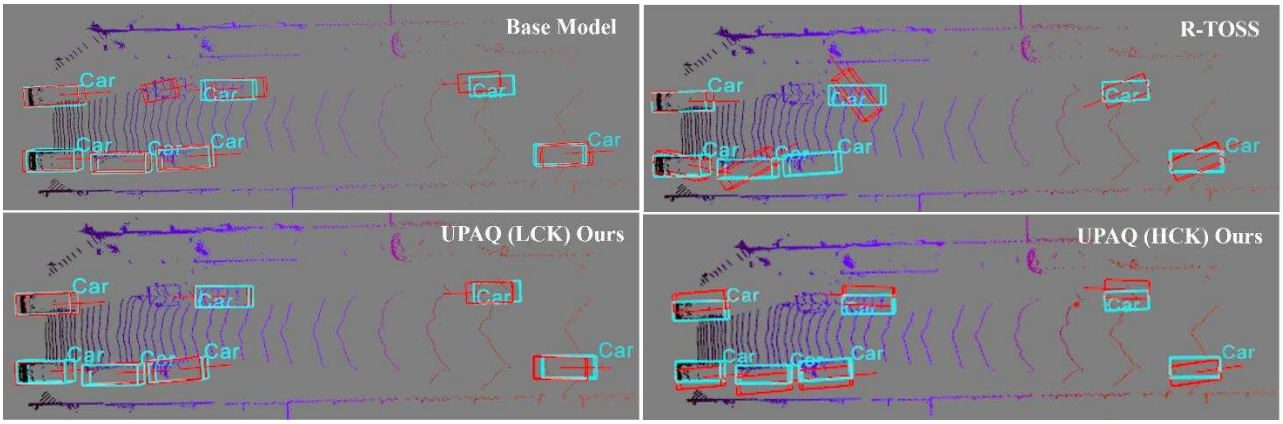


Figure 25 Comparison of output achieved in PointPillars model after using various compression frameworks. Blue bounding boxes indicate the ground truth positions of cars, while red boxes show each frameworks predictions

3.6 CONCLUSIONS

In this chapter, we introduced *UPAQ*, a novel 3D object detection (ODs) compression framework that aims to preserve model accuracy while significantly reducing storage requirements and computational (performance, energy) overheads. Our comprehensive experimental results and

comparative analyses demonstrate that *UPAQ* consistently outperforms state-of-the-art frameworks in terms of compression ratios, inference speed, and energy efficiency, while achieving a notable increase in mAP compared to baseline 3D ODs models for LiDAR and camera data. The framework effectively minimizes computational costs during both compression and inference, facilitating a more efficient model compression process. Specifically, our experiments on the Jetson Orin platform show that *UPAQ* achieves model compression rates of 5.62× for the PointPillars 3D ODs model and 5.13× for the SMOKE 3D ODs model, while also surpassing the original model in both mAP and inference speed, underscoring the efficacy of our approach. Our ongoing work is looking at combining deep learning techniques for anomaly detection [82][103] and sensor deployment [105][136] with optimized object detection.

4 CONCLUSION AND FUTURE WORK SUGGESTIONS

4.1 RESEARCH CONCLUSION

In this thesis, we introduce two innovative machine learning model optimization frameworks designed to reduce model footprint and accelerate inference on embedded autonomous platforms. These frameworks aim to enhance the efficiency of machine learning models, particularly in resource-constrained environments. To our knowledge, no previous research in academia or industry has applied kernel-based optimization in the manner we have to enhance machine learning models. Our approach uniquely leverages kernel-based optimization techniques to optimize models for faster inference, lower energy consumption, and improved accuracy.

Our research demonstrates that kernel-based optimization can significantly enhance model performance. Specifically, it enables reductions in both inference time and energy usage, which are crucial for embedded systems with limited resources. Additionally, our methods maintain or even improve model accuracy, ensuring that optimizations do not compromise performance. Furthermore, the reduction in model size makes these models more feasible for deployment on embedded platforms, where memory and storage are limited.

One of our major contributions is the introduction of a novel kernel-based semi-structured pruning technique called *R-TOSS* (Chapter 2). *R-TOSS* is a highly efficient framework for real-time object detection, particularly in applications like AVs where memory and computational overheads are critical concerns. Our proposed framework achieves significant improvements in terms of inference time and energy efficiency while reducing model footprint. Specifically, we achieved a compression rate of 4.4x on the YOLOv5 model and 2.89x on the RetinaNet model.

We observed a 2.15x improvement in inference speed and a 0.43x reduction in energy usage, along with improved model accuracy. These results demonstrate the great promise of *R-TOSS* in providing highly efficient and accelerated inference for object detection models. Unlike existing state-of-the-art pruning frameworks, *R-TOSS* can accelerate any existing variant of object detection models.

Our second contribution involves a unified pruning and quantization approach called *UPAQ*. *UPAQ* significantly contributes to real-time and energy-efficient 3D object detection in AVs. Unlike other state-of-the-art frameworks, *UPAQ* employs a combination of pruning and quantization techniques applied to a set of computationally significant layers, which are then propagated to layers that are part of the same computational chain. This approach decreases the computational cost of model compression while improving model performance in terms of inference speed, model compression, and energy efficiency for faster real-time inference. Specifically, *UPAQ* achieves at least a 5.13x reduction in model footprint, a 1.86x improvement in inference speed, and a 1.87x reduction in energy usage. These results signify the efficiency of *UPAQ* compared to other compression frameworks while maintaining the accuracy of the model.

In summary, the *R-TOSS* and *UPAQ* frameworks offer significant advancements in the optimization of machine learning models for embedded autonomous platforms, providing enhanced efficiency, faster inference, lower energy consumption, and improved model accuracy. These contributions highlight the potential of kernel-based optimization techniques in advancing the field of machine learning.

4.2 SUGGESTIONS FOR FUTURE WORK

While both compression techniques presented in this thesis, *R-TOSS* and *UPAQ*, demonstrated significant improvements in energy efficiency and throughput, the challenge of

designing efficient compression methods for object detection models will continue to grow in complexity and demand. As object detection models become more advanced and require increasingly sophisticated processing capabilities, further innovations in compression techniques are essential. These improvements are necessary to handle the ever-expanding size and complexity of deep neural networks (DNNs), while maintaining the speed and accuracy required for real-time performance in dynamic environments. Additionally, as object detection finds applications across industries like autonomous driving, security, and robotics, enhancing compression methods will be crucial for reducing computational costs and ensuring the efficiency of these systems, without compromising detection quality.

4.2.1 *Neural Architecture Search (NAS) for Object Detection:*

Recent efforts in NAS (Neural Architecture Search) have attracted attention for automating the design of object detection models, traditionally a time-consuming and manual process. NAS techniques can help identify optimal architectures, improving the performance of ODs in tasks like image classification. Works such as NAS-FCOS [137], MobileDets [138], and AutoDets [139] have shown the potential of NAS to enhance detection accuracy and efficiency by optimizing anchor boxes and backbone networks. However, the process of NAS is computationally expensive and time-consuming, often requiring significant resources to explore the large architecture space. As such, more efficient NAS methods are needed to reduce the time and computational cost while maintaining model performance. Future research should aim to streamline NAS for ODs, potentially integrating model compression techniques presented in chapters 2 and 3 to further enhance efficiency and reduce search space complexity.

4.2.2 Time Series Information for Enhanced Object Detection:

Most traditional object detection models rely on CNNs, which analyze frames in isolation, missing valuable temporal context. Incorporating time-series information from multiple frames can improve object detection accuracy by capturing the motion and dynamics of objects over time. Research like Sauer et al. [141] and Chen et al. [142] has shown that integrating vehicle dynamics such as steering angles and velocities enhances the performance of object detection systems, especially in autonomous driving. By combining multi-frame perception with time-series data, object detection models can make better driving decisions, improving safety and responsiveness. Future advancements in temporal context integration, possibly in combination with compression techniques presented in chapters 2 and 3, could further improve model efficiency and enable real-time processing for AVs.

4.2.3 Semi-Supervised Object Detection for Autonomous Vehicles:

Supervised machine learning requires annotated datasets, which are time-consuming to collect, especially in varying real-world scenarios. Semi-supervised learning offers a potential solution by reducing the need for extensive annotations while still delivering high-performance ODs. Transformer-based models, known for their accuracy, have shown promise in semi-supervised settings for autonomous driving [143]-[145]. However, the challenge remains in deploying these large models on embedded systems due to their substantial memory requirements. Optimizing these models through techniques presented in chapters 2 and 3 can alleviate memory constraints and make them more practical for real-world applications in AVs, where resources are limited.

4.2.4 Synthesis-Based Object Detection:

Synthesis-based object detection [146] has gained attention as it overcomes the reliance on large, labeled real-world datasets. By generating synthetic data through techniques like GANs and NeRFs, researchers can create diverse datasets under various environmental conditions and object variations. This approach helps train robust object detection models capable of generalizing across different scenarios. However, a significant challenge is the domain gap between synthetic and real-world data, which often leads to reduced model performance when transitioning from synthetic to real environments. Approaches like adversarial training and style transfer, combined with model compression techniques such as *R-TOSS* and *UPAQ*, can help bridge this gap. Furthermore, integrating physics-based simulations with generative models could improve the realism of synthetic data, especially for applications in robotics and autonomous systems. Real-time synthesis pipelines that generate data on-the-fly are also a promising direction for continual learning, enabling systems to adapt to new, dynamic environments without forgetting prior knowledge.

4.2.5 Sensor Deployment for Object Detection:

Sensor deployment plays a critical role in the performance of object detection systems, particularly in applications like autonomous driving, robotics, and security. The choice and placement of sensors, such as LiDAR, cameras, and radar, significantly impact the accuracy and robustness of object detection models. Proper sensor deployment ensures that the system can capture diverse environmental conditions, including varying light levels, weather conditions, and different object distances. Recent work by [147][105][136] emphasizes the importance of sensor fusion, where data from multiple sensors are integrated to improve the reliability and accuracy of object detection. In autonomous driving, for example, the combination of LiDAR and camera data provides a more complete understanding of the vehicle's surroundings, enabling better decision-

making. Research into optimal sensor deployment strategies, such as sensor placement algorithms and sensor fusion techniques, is crucial for enhancing object detection system performance in real-world scenarios. Advances in model compression techniques like *R-TOSS* and *UPAQ* can help mitigate the computational challenges posed by large-scale sensor data, allowing for more efficient sensor deployment in autonomous systems.

4.2.6 Anomaly Detection for Object Detection Systems:

Anomaly detection is essential for improving the reliability and safety of object detection systems, particularly in dynamic environments such as autonomous driving, industrial monitoring, and security applications. Anomalies, which can include unexpected behaviors or rare events, may indicate critical situations requiring immediate attention. Traditional object detection models often struggle to identify such anomalies due to the limited variety in training data. Recent approaches, like those in [82][103], combine deep learning techniques with anomaly detection algorithms to detect unusual patterns in real-time sensor data, such as sudden object movements or environmental changes. By integrating anomaly detection into object detection pipelines, systems can automatically flag unusual scenarios, allowing for faster response times and better decision-making. This integration becomes particularly crucial for AVs, where detecting unexpected road conditions or pedestrian behaviors can prevent accidents. Additionally, the application of model compression techniques presented in chapters 2 and 3 can reduce the computational overhead of anomaly detection processes, enabling real-time processing and more efficient deployment of these systems in resource-constrained environments.

BIBLIOGRAPHY

[1] R. A. Acheampong, F. Cugurullo, I. Dusparic, and M. Gueriau, "Can autonomous vehicles enable sustainable mobility in future cities? Insights and policy challenges from user preferences over different urban transport options," *Cities, vol. 112, Article 103134, 2021*.

[2] M. Kovačić, M. Mutavdžija, and K. Buntak, "New Paradigm of Sustainable Urban Mobility: Electric and Autonomous Vehicles—A Review and Bibliometric Analysis," *Sustainability,* vol. 14, no. 15, pp. 9525, 2022.

[3] E.A. Thompson, E. T. Akoto, H. B. Atuobi, P. Lu, and C. K. Abbew, "Synergizing Urban Mobility: The Interplay between Autonomous Vehicles and Autonomous Parking Spaces for Sustainable Development," *Journal of Transportation Technologies,* 15, no. 1, pp. 50-59, 2024.

[4] Statista, "Collisions & Crashes Per Vehicle Miles Traveled by Type of Vehicle," Available: *https://www.statista.com/statistics/1234567/collisions-crashes-per-vehicle-miles-traveled-by-type-of-vehicle/*. [Accessed on: Mar. 7, 2025].

[5] World Economic Forum, "How Will Autonomous Vehicles Shape Urban Mobility?" Available: *https://www.weforum.org/how-will-autonomous-vehicles-shape-urban-mobility*. [Accessed on: Mar. 7, 2025].

[6] Consumer Affairs, "Autonomous Vehicle Safety Statistics," Available: *https://www.consumeraffairs.com/autonomous-vehicle-safety-statistics*. [Accessed on: Mar. 7, 2025].

[7] McKinsey, "Autonomous Driving's Future: Convenient and Connected," Available: *https://www.mckinsey.com/autonomous-driving-future*. [Accessed on: Mar. 7, 2025].

[8] J. R. Smith, M. L. Jones, and A. B. Taylor, “Rethinking Advanced Driver Assistance System taxonomies: A framework and inventory of real-world safety performance*,” IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 7, pp. 1234-1245, 2023.

[9] M. Reda, A. Onsy, A. Y. Haikal, and A. Ghanbari, “Path planning algorithms in the autonomous driving system: A comprehensive review,” *Robotics and Autonomous Systems,* vol. 174, pp. 104630, 2024.

[10] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, “Multi-view 3D object detection network for autonomous driving,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1907-1915. 2017.

[11] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2005, pp. 886-893*.

[12] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI),* vol. 39, no. 6, pp. 1137-1149, 2017.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg, “SSD: Single shot multibox detector,” *European Conference on Computer Vision (ECCV),* 2016.

[14] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracker,” *IEEE International Conference on Image Processing (ICIP),* pp. 3464-3468, 2016.

[15] A. Sadat, M. Ren, A. Pokrovsky, S. Yun, and R. Urtasun, “Jointly learnable behavior and trajectory planning for self-driving vehicles,” *IEEE International Conference on Robotics and Automation (ICRA),* pp. 3315-3321. 2016.

[16] National Highway Traffic Safety Administration (NHTSA), "Automated Vehicles for Safety," *NHTSA Report, 2021*. Available: *https://www.nhtsa.gov/automated-vehicles-safety*. [Accessed on: Mar. 7, 2025].

[17] V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley, "Advanced Driver-Assistance Systems: A Path Toward Autonomous Vehicles," *IEEE Consumer Electronics Magazine,* vol. 7, no. 5, pp. 18-25, 2018.

[18] SAE International - Mobility Engineering, SAE International, 2018. Available: *https://www.sae.org/standards/content/j3016_201806*. [Accessed: Mar. 7, 2025].

[19] R. D. Staff, "Navlab: The self-driving car of the '80s," Rediscover the '80s, 2016. Available: *https://rediscover-the-80s.com/navlab-self-driving-car*. [Accessed: Mar. 7, 2025].

[20] E. D. Dickmanns, "Dynamic Vision for perception and control of Motion", *Springer 2010*.

[21] R. Lawler, "Riding a shotgun in Tesla's fastest car ever", *Engadget*, 2014, Available: *https://tinyurl.com/2a65z9nz*. [Accessed: Mar. 7, 2025].

[22] "Drive Me, the world's most ambitious and advanced public autonomous driving experiment, starts today," *Volvo Cars Global Media Newsroom, 2016*. Available: *https://www.media.volvocars.com/global/en-gb/media/pressreleases/177072*. [Accessed: Mar. 7, 2025].

[23] Safety report and Whitepapers, *Waymo*, Available: *https://waymo.com/safety/*. [Accessed: Mar. 7, 2025].

[24] S. McEachern, M. Smith, J. Doe, and L. Johnson, "Cruise founder takes company's first driverless ride on SF streets: Video," *GM Authority, Nov. 2021*. Available: *https://gmauthority.com/blog/2021/11/cruise-founder-takes-companys-first-driverless-ride-on-sf-streets-video/*. [Accessed: Mar. 7, 2025].

[25] L. Jiao, X. Liu, Z. Zhang, and Y. Liu, "A survey of deep learning-based object detection," *IEEE Access,* vol. 7, pp. 128837-128868, 2019.

[26] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 886-893, vol. 1, 2005.

[27] P. F. Felzenszwalb, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *in IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 32, no. 9, pp. 1627-1645, 2010.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 580-587, 2014.

[29] R. Girshick, "Fast R-CNN," *In Proceedings of the IEEE international conference on computer vision,* pp. 1440-1448. 2015.

[30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence,* vol. 39, no. 6, pp. 1137-1149, 2017.

[31] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," In *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 779-788. 2016.

[32] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *In Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 7263-7271, 2017.

[33] J. Redmon, A. Farhadi, and C. B. B. H. Yang, "YOLOv3: An incremental improvement," *arXiv:1804.02767,* 2018.

[34] W. Liu, D. Anguelov, D. Erhan, et al., "SSD: Single Shot MultiBox Detector," *European Conference on Computer Vision,* Springer, Cham, 2016.

[35] T.-Y. Lin, P. G. M. Zisserman, M. R. Girshick, et al., "Focal loss for dense object detection," In *proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 2980-2988, 2017.

[36] A. Bochkovskiy, C. Y. Wang, and H. P. B. G. A. C. M. G., "YOLOv4: Optimal speed and accuracy of object detection," *arXiv:2004.10934, 2020*.

[37] Ultralytics, "Ultralytics/yolov5: Yolov5 in PyTorch & ONNX & CoreML & TFLite," GitHub. Available: *https://github.com/ultralytics/yolov5*. [Accessed: Mar. 7, 2025].

[38] C.Y. Wang, H. H. Chen, and Y. H. Li, "You Only Learn One Representation: Unified Network for Multiple Tasks," *arXiv preprint arXiv:2105.04206, 2021*.

[39] Z. Ge, H. Zhuang, X. Luo, and M. Li, "Yolox: Exceeding Yolo Series in 2021," *arXiv preprint arXiv:2107.08430, 2021*.

[40] R. team, "YOLO-NAS by Deci achieves state-of-the-art performance on object detection using neural architecture search," *Deci.ai Blog*, *2023*. Available: *https://deci.ai/blog/yolo-nas-object-detection-foundation-model.* [Accessed: Mar. 7, 2025].

[41] R. Khanam and M. Hussain, "YOLOv11: An Overview of the Key Architectural Enhancements," *arXiv preprint arXiv:2410.17725,* 2024.

[42] X. Hou, M. Liu, S. Zhang, P. Wei, B. Chen, and X. Lan, "Relation DETR: Exploring Explicit Position Relation Prior for Object Detection," in *Proceedings of the European Conference on Computer Vision (ECCV),* 2024.

[43] Y. Tian, Q. Ye, and D. Doermann, "YOLOv12: An Overview of the Key Architectural Enhancements," *GitHub, 2025*. Available: *https://github.com/sunsmarterjie/yolov12*. [Accessed: Mar. 7, 2025].

[44] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection," *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF),* pp. 1-7, 2019.

[45] J. Fang, L. Zhou, and G. Liu, "3D Bounding Box Estimation for Autonomous Vehicles by Cascaded Geometric Constraints and Depurated 2D Detections Using 3D Results," *arXiv preprint arXiv:1909.01867,* 2019.

[46] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: Real-time 3D object detection on point clouds," *arXiv preprint arXiv:1803.06199,* 2018.

[47] M. Simon, K. Amende, A. Kraus, J. Honer, T. Sämann, H. Kaulbersch, S. Milz, and H. M. Gross, "Complexer-YOLO: Real-time 3D object detection and tracking on semantic point clouds," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops,* pp. 0-0. 2019.

[48] L. H. Wen and K. H. Jo, "Fast and Accurate 3D Object Detection for Lidar-Camera-Based Autonomous Vehicles Using One Shared Voxel-Based Backbone," *in IEEE Access,* vol. 9, pp. 22080-22089, 2021.

[49] Y. Lu, X. Hao, Y. Li, W. Chai, S. Sun, and S. Velipasalar, "Range-aware attention network for lidar-based 3D object detection with auxiliary point density level estimation," *IEEE Transactions on Vehicular Technology*, 2024.

[50] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," *arXiv preprint arXiv:1812.04244, 2018.*

[51] Q. Cai, Y. Pan, T. Yao, C.-W. Ngo, and T. M., "ObjectFusion: Multi-modal 3D object detection with object-centric fusion," *In Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18067-18076. 2023.

[52] S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, J. Shi, X. Wang, and H. Li, "PV-RCNN++: Point-voxel feature set abstraction with local vector representation for 3D object detection," *arXiv preprint*, 2021.

[53] L. Jia, R. Guan, H. Zhao, Q. Zhao, K. L. Man, J. Smith, L. Yu, and Y. Yue, "RadarNeXt: Real-time and reliable 3D object detector based on 4D mmWave imaging radar," *arXiv preprint*, 2023.

[54] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *arXiv preprint*, 2023.

[55] Q. Wang, H. Zhang, X. Hong, and Q. Zhou, "Small object detection based on modified FSSD and model compression," *In Proceedings of the IEEE 6th International Conference on Signal and Image Processing (ICSIP),* pp. 88-92, 2021.

[56] P. Zhao, X. Zhang, Y. Liu, Z. Wang, and H. Li, "Neural pruning search for real-time object detection of autonomous vehicles," In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 835-840. IEEE, 2021.

[57] H. Fan, S. Liu, M. Ferianc, H. C. Ng, Z. Que, S. Liu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," In *Proceedings of the 2018 International conference on field-programmable technology (FPT)*, pp. 14-21. IEEE, 2018.

[58] S. Tripathi, G. Dane, B. Kang, V. Bhaskaran, and T. Nguyen, "LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems,"

In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 94-103. 2017.

[59] Z. Kang, P. Zhang, X. Zhang, J. Sun, and N. Zheng, "Instance-Conditional Knowledge Distillation for Object Detection," *in Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[60] R. Chen, H. Ai, C. Shang, L. Chen, and Z. Zhuang, "Learning lightweight pedestrian detector with hierarchical knowledge distillation," In *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1645-1649. IEEE, 2019.

[61] Waymo, "Waymo Open Dataset: Autonomous Driving Dataset," Waymo, 2020, Available: https://waymo.com/open. [Accessed: Mar. 7, 2025].

[62] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354-3361, 2012.

[63] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A Multimodal Dataset for Autonomous Driving," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (CVPR), pp. 11621-11631. 2020.

[64] M. F. Chang, J. W. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3D Tracking and Forecasting with Rich Maps," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (CVPR), 2019

[65] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The ApolloScape Dataset for Autonomous Driving," In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 954-960. 2018.

[66] M. Enzweiler and D. M. Gavrila, "Monocular Pedestrian Detection: Survey and Experiments," IEEE *transactions on pattern analysis and machine intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.

[67] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning," *arXiv, 1805.04687,* 2018.

[68] H. Zhu, H. Wei, B. Li, X. Yuan, and N. Kehtarnavaz, "Real-Time Moving Object Detection in High-Resolution Video Sensing," *Sensors*, vol. 20, no. 12, p. 3591, 2020.

[69] X. Dai, X. Yuan, and X. Wei, "TIRNet: Object detection in thermal infrared images for autonomous driving," *Applied Intelligence*, vol. 51, no. 3, pp. 1244–1261, Mar. 2021.

[70] R. O. Chávez-García and O. Aycard, "Multiple Sensor Fusion and Classification for Moving Object Detection and Tracking," *IEEE Transactions on Intelligent Transportation Systems 17*, 2015.

[71] H. Cho, Y. W. Seo, B. V. Kumar, and R. R. Rajkumar, "A multi-sensor fusion system for moving object detection and tracking in urban driving environments," In *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 1836-1843. IEEE, 2014.

[72] P. Chen, J. Liu, B. Zhuang, M. Tan, and C. Shen, "AQD: Towards accurate quantized object detection," In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 104-113. 202.

[73] S. Kim and H. Kim, "Zero-centered fixed-point quantization with iterative retraining for deep convolutional neural network-based object detectors," *IEEE Access*, vol. 9, 2021.

[74] R. Pilipović, V. Risojević, J. Božič, P. Bulić, and U. Lotrič, "An approximate GEMM unit for energy-efficient object detection," *Sensors*, vol. 21, no. 12, 2021.

[75] S. Winograd, Arithmetic complexity of computations, vol. 33, Siam, 1980.

[76] S. Kala, J. Mathew, B. R. Jose, and S. Nalesh, "UniWiG: Unified Winograd-GEMM architecture for accelerating CNN on FPGAs," In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pp. 209-214. IEEE, 2019.

[77] X. Zhang, H. Lu, C. Hao, J. Li, B. Cheng, Y. Li, K. Rupnow, et al., "SkyNet: a hardware-efficient method for object detection and tracking on embedded systems," *In Proceedings of Machine Learning and Systems*, vol. 2, 2020.

[78] Y. Zhu, Y. Liu, D. Zhang, S. Li, P. Zhang, and T. Hadley, "Acceleration of pedestrian detection algorithm on novel C2RTL HW/SW co-design platform," In *The 2010 International Conference on Green Circuits and Systems*, pp. 615-620. IEEE, 2010.

[79] Y. Ma, T. Zheng, Y. Cao, S. Vrudhula, and J.S. Seo, "Algorithm-Hardware Co-Design of Single Shot Detector for Fast Object Detection on FPGAs", in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-8. IEEE, 2018.

[80] Automated Driving Systems, NHTSA. Available: https://www.nhtsa.gov/. [Accessed: Mar. 8, 2025].

[81] S. Mandal, S. Biswas, V. E. Balas, R. N. Shaw, and A. Ghosh, "Lyft 3D object detection for autonomous vehicles," *In Artificial Intelligence for Future Generation Robotics, Elsevier*, 2021.

[82] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, "Roadmap for Cybersecurity in Autonomous Vehicles", *IEEE Consumer Electronics Magazine*, Vol. 11, Iss. 6, pp. 13-23, IEEE Consumer Electronics, Nov 2022.

[83] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, 2020.

[84] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574,* 2019.

[85] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, and D. Alistarh, "Inducing and exploiting activation sparsity for fast inference on deep neural networks," *In International Conference on Machine Learning*, vol. 119, PMLR, Nov. 2020.

[86] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," *In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition,* pp. 11264-11272. 2019.

[87] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[88] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *Advances in Neural Information Processing Systems*, vol. 332020.

[89] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," arXiv preprint arXiv:2002.07376, 2020.

[90] V. Crescitelli, S. Miura, G. Ono, and N. Kohmu, *"*Edge devices object detection by filter pruning,*" In 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021.

[91] D. Filters'Importance, *"*Pruning filters for efficient ConvNets,*"* 2016.

[92] H. Wang, C. Qin, Y. Zhang, and Y. Fu, *"*Neural pruning via growing regularization,*" arXiv preprint arXiv:2012.09243*, 2020.

[93] Z. Xie, L. Zhu, L. Zhao, B. Tao, L. Liu, and W. Tao, *"*Localization-aware channel pruning for object detection,*" Neurocomputing*, vol. 403, 2020.

[94] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, *"*Learning efficient convolutional networks through network slimming,*" In Proceedings of the IEEE International Conference On Computer Vision*, 2017.

[95] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, and C. L. Zitnick, *"*Microsoft COCO: Common objects in context,*"* In *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, pp. 740-755. Springer International Publishing, 2014.

[96] E. Jeong, J. Kim, and S. Ha, *"*Tensorrt-based framework and optimization methodology for deep learning inference on Jetson boards,*" ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 5, 2022.

[97] V. Kukkala, S. Pasricha, T. H. Bradley, *"*SEDAN: Security-Aware Design of Time-Critical Automotive Networks*"*, *IEEE Transactions on Vehicular Technology (TVT)*, vol. 69, no. 8, 2020.

[98] J. Pool, *"*Accelerating sparsity in the Nvidia Ampere architecture,*"* Available on: *https://tinyurl.com/4439phxe* [Accessed: Mar. 7, 2025].

[99] F. Sunny, M. Nikdast, and S. Pasricha, *"*SONIC: A Sparse Neural Network Inference Accelerator with Silicon Photonics for Energy-Efficient Deep Learning,*"* In *2022 27th asia and south pacific design automation conference (ASP-DAC)*, pp. 214-219. IEEE, 2022.

[100] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, and B. Ren, *"*PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning,*"* In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 907-922. 2020.

[101] Y. Cai, H. Li, G. Yuan, W. Niu, Y. Li, X. Tang, and Y. Wang, *"*Yolobile: Real-time object detection on mobile devices via compression-compilation co-design,*"* In *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 2, pp. 955-963. 2021.

[102] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *"*End-to-end object detection with transformers,*"* In *European conference on computer vision*, pp. 213-229. Cham: Springer International Publishing, 2020.

[103] V. K. Kukkala, S. V. Thiruloga, and S. Pasricha, *"*LATTE: LSTM Self-Attention based Anomaly Detection in Embedded Automotive Platforms*"*, *IEEE/ACM CODES+ISSS (ESWEEK)*, 2021.

[104] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[105] J. Dey, W. Taylor, S. Pasricha, *"*VESPA: Optimizing Heterogeneous Sensor Placement and Orientation for Autonomous Vehicles*"*, *IEEE Consumer Electronics, 10(2)*, 2021.

[106] NVIDIA Corporation, *"*GeForce RTX 20 Series Graphics Cards,*"* NVIDIA, Available: *https://www.nvidia.com/en-us/geforce/20-series/*. [Accessed: Mar. 7, 2025]

[107] Jetson TX2 Module; Available on: *https://elinux.org/Jetson_TX2* [Accessed: Mar. 7, 2025]

[108] J. Dey, S. Pasricha, *"*Robust Perception Architecture Design for Automotive Cyber-Physical Systems*"*, *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022.

[109] A. Balasubramaniam, S. Pasricha, *"*Object detection in autonomous vehicles: Status and open challenges.*" arXiv preprint arXiv:2201.07706*, 2022.

[110] V. Kukkala, S. Pasricha, *"*Machine Learning and Optimization Techniques for Automotive Cyber-Physical Systems*"*, *Springer Nature Publishers*, 2023.

[111] Q. He, A. Xu, Z. Ye, W. Zhou, and T. Cai, *"*Object detection based on lightweight YOLOX for autonomous driving,*" Sensors*, vol. 23, no. 17, 2023.

[112] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, *"*Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks,*" Journal of Machine Learning Research*, vol. 22, no. 241, 2021.

[113] A. Kuzmin, M. Nagel, M. Van Baalen, A. Behboodi, and T. Blankevoort, *"*Pruning vs quantization: Which is better?,*" Advances in Neural Information Processing Systems*, vol. 36, 2023.

[114] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez, *"*On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data,*" Remote Sensing*, vol. 13, no. 1, 2020.

[115] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *"*PointNet: Deep learning on point sets for 3D classification and segmentation,*"* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[116] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[117] Y. Zhang, J. Lu, and J. Zhou, "Objects are different: Flexible monocular 3D object detection," In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[118] Z. Liu, Z. Wu, and R. Tóth, "SMOKE: Single-stage monocular 3D object detection via keypoint estimation," In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[119] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[120] Y. Chen, Y. Li, X. Zhang, J. Sun, and J. Jia, "Focal sparse convolutional networks for 3D object detection," In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[121] H. Wu, C. Wen, S. Shi, X. Li, and C. Wang, "Virtual sparse convolution for multimodal 3D object detection," In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[122] B. Hawks, J. Duarte, N. J. Fraser, A. Pappalardo, N. Tran, and Y. Umuroglu, "Ps and Qs: Quantization-aware pruning for efficient low latency neural network inference," *Frontiers in Artificial Intelligence*, vol. 4, 2021.

[123] F. Tung and G. Mori, "Clip-Q: Deep network compression learning by in-parallel pruning-quantization," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[124] S. Zhou, L. Li, X. Zhang, B. Zhang, S. Bai, M. Sun, and X. Chu, *"*LiDAR-PTQ: Post-training quantization for point cloud 3D object detection,*" arXiv preprint arXiv:2401.15865*, 2024.

[125] A. Balasubramaniam, F. Sunny, S. Pasricha, *"*R-TOSS: A framework for real-time object detection using semi-structured pruning.*" 2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.

[126] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, and D. Alistarh, *"*Inducing and exploiting activation sparsity for fast inference on deep neural networks,*"* In *International Conference on Machine Learning*, 2020.

[127] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, *"*Importance estimation for neural network pruning,*"* In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[128] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, *"*Pruning neural networks without any data by iteratively conserving synaptic flow,*" Advances in Neural Information Processing Systems*, vol. 33, 2020.

[129] V. Crescitelli, S. Miura, G. Ono, and N. Kohmu, *"*Edge devices object detection by filter pruning,*"* In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021.

[130] Z. Xie, L. Zhu, L. Zhao, B. Tao, L. Liu, and W. Tao, *"*Localization-aware channel pruning for object detection,*" Neurocomputing*, vol. 403, 2020.

[131] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, *"*Loss aware post-training quantization,*" Machine Learning*, vol. 110, no. 11, pp. 3245–3262, 2021.

[132] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, *"*A survey of quantization methods for efficient neural network inference,*"* In *Low-Power Computer Vision, Chapman and Hall/CRC*, 2022.

[133] Shallow and deep copy operations - Python documentation,*"* *docs.python.org.* Available: *https://docs.python.org/3/library/copy.html* [Accessed: Mar. 7, 2025]

[134] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, and D. Kalenichenko, *"*Quantization and training of neural networks for efficient integer-arithmetic-only inference,*"* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[135] wildkid1024, *"*NVpower at master · wildkid1024/NVpower,*"* *GitHub*, 2020. Available: *https://github.com/wildkid1024/NVpower/tree/master/NVpower* [Accessed: Mar. 7, 2025]

[136] J. Dey, S. Pasricha, *"*Co-Optimizing Sensing and Deep Machine Learning in Automotive Cyber-Physical Systems*"*, *IEEE Euromicro Conference on Digital Systems Design*, 2022.

[137] N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, and Y. Zhang, *"*NAS-FCOS: Efficient search for object detection architectures,*"* *International Journal of Computer Vision*, vol. 129, 2021.

[138] Y. Xiong, H. Liu, S. Gupta, B. Akin, G. Bender, Y. Wang, and B. Chen, *"*MobileDets: Searching for object detection architectures for mobile accelerators,*"* In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[139] Z. Li, T. Xi, G. Zhang, J. Liu, and R. He, *"*AutoDet: Pyramid network architecture search for object detection,*"* *International Journal of Computer Vision*, vol. 129, 2021.

[140] S. Casas, W. Luo, and R. Urtasun, *"*IntentNet: Learning to predict intention from raw sensor data,*"* In *Conference on Robot Learning*, 2018.

[141] A. Sauer, N. Savinov, and A. Geiger, *"*Conditional affordance learning for driving in urban environments,*"* In *Conference on Robot Learning*, 2018.

[142] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, *"*DeepDriving: Learning affordance for direct perception in autonomous driving,*"* In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[143] E. Xie, J. Ding, W. Wang, X. Zhan, H. Xu, P. Sun, and P. Luo, *"*DetCo: Unsupervised contrastive learning for object detection,*"* In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[144] Z. Dai, B. Cai, Y. Lin, and J. Chen, *"*UP-DETR: Unsupervised pre-training for object detection with transformers,*"* In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[145] A. Bar, X. Wang, V. Kantorov, C. J. Reed, R. Herzig, G. Chechik, and A. Globerson, *"*DetReg: Unsupervised pretraining with region priors for object detection,*"* In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14605-14615, 2022.

[146] S. K. Mustikovela, S. De Mello, A. Prakash, U. Iqbal, S. Liu, T. Nguyen-Phuoc, and J. Kautz, *"*Self-supervised object detection via generative image synthesis,*"* In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[147] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, *"*Sensor and sensor fusion technology in autonomous vehicles: A review,*"* *Sensors*, vol. 21, no. 6, 2021.